GaussDB
8.x

Developer Guide for Primary +Standby Instances

Issue	01
Date	2024-05-31

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei Cloud and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Cloud Computing Technologies Co., Ltd.

Address: Huawei Cloud Data Center Jiaoxinggong Road
Qianzhong Avenue
Gui'an New District
Gui Zhou 550029
People's Republic of China

Website: https://www.huaweicloud.com/intl/en-us/

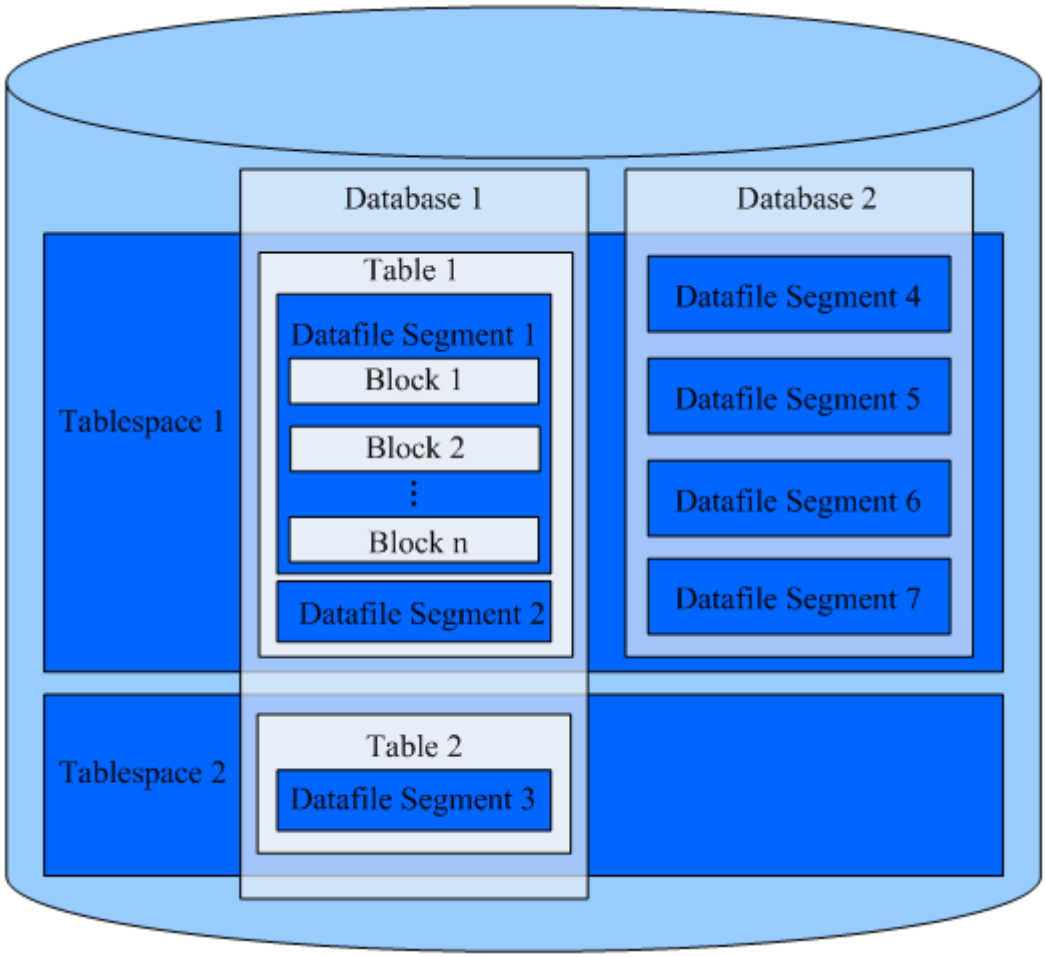
Contents

1 Database System Overview

1.1 Database Logical Architecture

Data nodes (DNs) in GaussDB store data on disks. This section describes the objects on each DN from the logical view and the relationship between these objects. **Figure 1-1** shows the database logical structure.

Figure 1-1 Database logical architecture

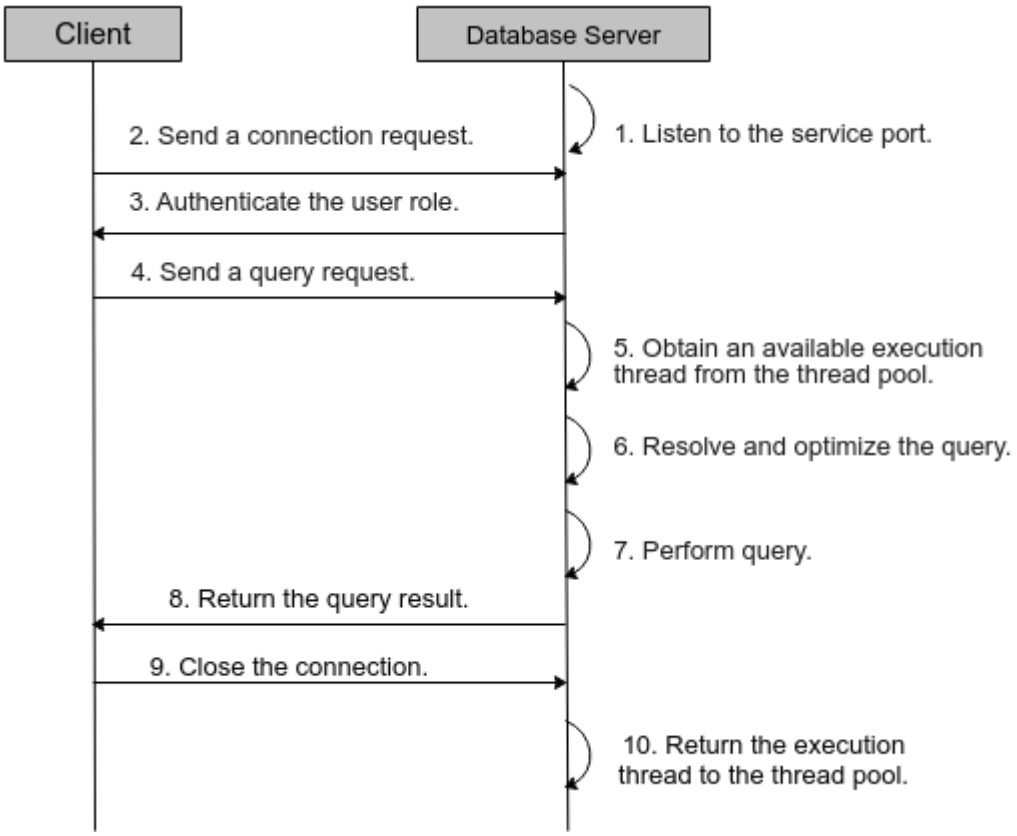


NOTE

- A tablespace is a directory. A database contains one or more tablespaces to store physical files of the database. Each tablespace can contain files belonging to different databases.
- A database manages various data objects and is isolated from each other. Objects managed by a database can be distributed to multiple tablespaces.
- A datafile segment stores data of only one table. A table containing more than 1 GB of data is stored in multiple datafile segments.
- One table belongs to only one database and one tablespace. The datafile segments storing the data of the same table must be in the same tablespace.
- Block: Basic unit of database management. Its default size is 8 KB.

1.2 Query Request Handling Process

Figure 1-2 GaussDB service response process



1.3 Managing Transactions

A transaction is a customized sequence of database operations, which form an integral unit of work. In GaussDB, you can start, set, commit, and roll back transactions. GaussDB supports the following transaction isolation levels: READ COMMITTED, REPEATABLE READ, and SERIALIZABLE. SERIALIZABLE is equivalent to REPEATABLE READ.

Controlling Transactions

The following describes transaction operations supported by the database:

- Starting transactions

  You can use the START TRANSACTION or BEGIN syntax to start a transaction. For details, see **START TRANSACTION** and **BEGIN**.

- Setting transactions

  You can use the SET TRANSACTION or SET LOCAL TRANSACTION syntax to set transactions. For details, see **SET TRANSACTION**.

- Committing transactions

You can commit all operations of a transaction using COMMIT or END. For details, see **COMMIT | END**.

- Rolling back transactions

  Rollback indicates that the system cancels all changes that a transaction has made to a database if the transaction fails to be executed due to a fault. For details, see **ROLLBACK**.

Transaction Isolation Levels

A transaction isolation level specifies how concurrent transactions process the same object.

NOTE

The isolation level cannot be changed after the first data manipulation statement (SELECT, INSERT, DELETE, UPDATE, FETCH, or COPY) in a transaction is executed.

- **READ COMMITTED**: A transaction can read only committed data. This is the default value.

  The SELECT statement accesses the snapshot of the database taken when the query begins. The SELECT statement can also access the data modifications in its transaction, regardless of whether they have been committed. Note that different database snapshots may be available to two consecutive SELECT statements for the same transaction, because data may be committed for other transactions while the first SELECT statement is executed.

  At the **READ COMMITTED** level, the execution of each statement begins with a new snapshot, which contains all the transactions that have been committed by the execution time. Therefore, during a transaction, a statement can access the result of other committed transactions. Check whether a single statement always accesses absolutely consistent data in a database.

  Transaction isolation at this level meets the requirements of many applications, and is fast and easy to use. However, applications performing complicated queries and updates may require data that is more consistent than this level can provide.

- **REPEATABLE READ**: At this level, a transaction can only read data committed before it starts. Uncommitted data or data committed in other concurrent transactions cannot be read. However, a query can read earlier data modifications in its transaction, regardless of whether they have been committed. **READ COMMITTED** differs from this level in that a transaction reads the snapshot taken at the start of the transaction, not at the beginning of the current query within the transaction. Therefore, the SELECT statement within a transaction always reads the same data, and cannot read data committed by other concurrent transactions after the transaction starts. Applications at this level must be able to retry transactions, because serialization failures may occur.
- **SERIALIZABLE**: Currently, GaussDB does not support this isolation level. Setting this isolation level is equivalent to REPEATABLE READ.

NOTE

REPEATABLE READ is implemented based on multi-version snapshots and write skew may occur. To avoid this scenario, perform the SELECT FOR UPDATE operation on the rows involved in the transaction. The following is an example of write skew:

Scenario 1: Table **a** has the **id** and **value** columns of the int type. Two data records are inserted. Assume that the sum of the values of the two data records must be less than or equal to 10 in the service logic of table **a**. Two transactions are concurrently started. The values are updated and modified based on the read values. After the modification, the sum of values is less than or equal to 10 in the transactions. After the modification is committed, the sum of values is 12, which violates the assumed service logic of table **a**.

```
gaussdb=# create table a(id int, value int);
CREATE TABLE
gaussdb=# insert into a values(1,4);
INSERT 0 1
gaussdb=# insert into a values(2,4);
INSERT 0 1
session1 :
gaussdb=# start transaction isolation level repeatable read;
START TRANSACTION
gaussdb=# select * from a;
id | value
----+-------
1 |     4
2 |     4
(2 rows)
gaussdb=# update a set value = 6 where id = 1;
UPDATE 1
gaussdb=# select * from a;
id | value
----+-------
1 |     6
2 |     4
(2 rows)
session2:
gaussdb=# start transaction isolation level repeatable read;
START TRANSACTION
gaussdb=# select * from a;
id | value
----+-------
1 |     4
2 |     4
(2 rows)
gaussdb=# update a set value = 6 where id = 2;
UPDATE 1
gaussdb=# select * from a;
id | value
----+-------
1 |     4
2 |     6
(2 rows)
session1:
gaussdb=# commit;
COMMIT
session2:
gaussdb=# commit;
COMMIT
gaussdb=# select * from a;
id | value
----+-------
1 |     6
2 |     6
(2 rows)
```


Scenario 2: Table **a** has the **id** and **value** columns of the int type. The **id** is the primary key. When the primary key is deleted and inserted concurrently, the values of two primary keys may be read in the transaction, violating the primary key constraint.

```
gaussdb=# create table a(id int primary key, value int);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "a_pkey" for table "a"
CREATE TABLE
gaussdb=# insert into a values(1,10);
INSERT 0 1
session1:
gaussdb=# start transaction isolation level repeatable read;
START TRANSACTION
gaussdb=# delete a where id = 1;
DELETE 1
session2:
gaussdb=# start transaction isolation level repeatable read;
START TRANSACTION
gaussdb=# select * from a;
id | value
----+-------
1 |    10
(1 row)
session1:
gaussdb=# commit;
COMMIT
session2:
gaussdb=# insert into a values(1, 100);
INSERT 0 1
gaussdb=# select * from a;
id | value
----+-------
1 |    10
1 |   100
(2 rows)
```

1.4 Concepts

Databases

Databases manage various data objects and are isolated from each other. While creating a database, you can specify a tablespace. If you do not specify it, the object will be saved to the PG_DEFAULT tablespace by default. Objects managed by a database can be distributed to multiple tablespaces.

Tablespaces

In GaussDB, a tablespace is a directory storing physical files of the databases. Multiple tablespaces can coexist. Files are physically isolated using tablespaces and managed by a file system.

Schemas

GaussDB schemas logically separate databases. All database objects are created under certain schemas. In GaussDB, schemas and users are loosely bound. When you create a user, a schema with the same name as the user will be created automatically. You can also create a schema or specify another schema.

Users and Roles

GaussDB uses users and roles to control the access to databases. A role can be a database user or a group of database users, depending on role settings. Each user

can have only one role. In GaussDB, the difference between roles and users is that a role does not have the LOGIN permission by default. In GaussDB, you can put a user's role under a parent role for flexible management.

Transactions

In GaussDB, transactions are managed by multi-version concurrency control (MVCC) and two-phase locking (2PL). It enables smooth data reads and writes. In GaussDB, an Astore engine saves historical version data together with the current tuple version. The GaussDB Astore engine uses a VACUUM thread instead of rollback segments to periodically delete historical version data. Generally, you do not need to pay special attention to the VACUUM thread unless you need to optimize the performance. The GaussDB Ustore engine stores historical version data to the undo rollback segments. The thread for purging undo deletes historical version data in a unified manner. In addition, GaussDB automatically commits transactions for single-statement queries (without using statements such as BEGIN to explicitly start a transaction block).

2 Database Security

2.1 Users and Permissions

2.1.1 Default Permission Mechanism

A user who creates an object is the owner of this object. By default, **separation of duties** is disabled after database installation. A database system administrator has the same permissions as object owners. After an object is created, only the object owner or system administrator can query, modify, and delete the object, and grant permissions for the object to other users through **GRANT** by default.

To enable another user to use the object, grant required permissions to the user or the role that contains the user.

GaussDB supports the following permissions: SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, CREATE, CONNECT, EXECUTE, USAGE, ALTER, DROP, COMMENT, INDEX, and VACUUM. Permission types are associated with object types. For permission details, see **GRANT**.

To revoke a permission that has been granted, see **REVOKE**. Object owners have implicit permissions (such as ALTER, DROP, COMMENT, INDEX, VACUUM, GRANT, and REVOKE) on objects. That is, once becoming the owner of an object, the owner is immediately granted the implicit permissions on the object. Object owners can remove their own common permissions (SELECT, INSERT, UPDATE, and DELETE), for example, making tables read-only to themselves or others, except the system administrator.

System catalogs and views are visible to either system administrators or all users. System catalogs and views that require system administrator permissions can be queried only by system administrators. For details, see **System Catalogs and System Views**.

The database provides the object isolation feature. If this feature is enabled, users can view only the objects (tables, views, columns, and functions) that they have the permission to access. System administrators are not affected by this feature. For details, see **ALTER DATABASE**.

It is not recommended to modify the permissions on system catalogs and system views.

2.1.2 Administrators

Initial User

The account automatically generated during database installation is called an initial user. The initial user is also the system administrator, monitor administrator, O&M administrator, and security policy administrator. It has the highest permissions in the system and can perform all operations. If the initial username is not specified during installation, the username is the same as the name of the OS user who installs the database. If the password of the initial user is not set during the installation, the password is empty after the installation. In this case, you need to change the password of the initial user on the gsql client before performing other operations. If the initial user password is empty, you cannot perform other SQL operations, such as upgrade, capacity expansion, and node replacement, except changing the password.

NOTE

- The OID of the initial user is 10, which can be queried in the pg_roles view.
- An initial user bypasses all permission checks. It is recommended that this user be used only as a database administrator for database management instead of service applications.

System Administrator

A system administrator is an account with the **SYSADMIN** attribute. By default, a system administrator has the same permissions as the object owner but does not have the object permissions in the dbe_perf schema.

To create a system administrator, connect to the database as the initial user or a system administrator and use the **CREATE USER** or **ALTER USER** statement with the **SYSADMIN** option.

```
gaussdb=# CREATE USER sysadmin WITH SYSADMIN password "********";
```

Or

```
gaussdb=# ALTER USER joe SYSADMIN;
```

To run the **ALTER USER** statement, the user must exist.

Security Administrator

A security administrator is an account with the **CREATEROLE** attribute. It has the permission to create, modify, and delete users or roles, and grant or revoke the permission of any non-system administrator, built-in role, permanent user, or O&M administrator.

If you want to create a security administrator and separation of duties is disabled, connect to the database as a system administrator or security administrator. If separation of duties is enabled, connect to the database as a security administrator and use the **CREATE USER** or **ALTER USER** statement with the **CREATEROLE** option.


```
gaussdb=# CREATE USER createrole WITH CREATEROLE password "********";
```

Or

```
gaussdb=# ALTER USER joe CREATEROLE;
```

To run the **ALTER USER** statement, the user must exist.

Audit Administrator

An audit administrator is an account with the **AUDITADMIN** attribute, which has the permission to view and delete audit logs.

If you want to create an audit administrator and separation of duties is disabled, connect to the database as a system administrator or security administrator. If separation of duties is enabled, connect to the database only as the initial user and use the **CREATE USER** or **ALTER USER** statement with the **AUDITADMIN** option.

```
gaussdb=# CREATE USER auditadmin WITH AUDITADMIN password "********";
```

Or

```
gaussdb=# ALTER USER joe AUDITADMIN;
```

To run the **ALTER USER** statement, the user must exist.

Monitor Administrator

A monitor administrator is an account with the **MONADMIN** attribute and has the permission to view views and functions in the dbe_perf schema. The monitor administrator can also grant or revoke object permissions in the dbe_perf schema.

To create a monitor administrator, connect to the database as a system administrator and use the **CREATE USER** or **ALTER USER** statement with the **MONADMIN** option.

```
gaussdb=# CREATE USER monadmin WITH MONADMIN password "********";
```

or

```
gaussdb=# ALTER USER joe MONADMIN;
```

To run the **ALTER USER** statement, the user must exist.

O&M Administrator

An O&M administrator is an account with the **OPRADMIN** attribute and has the permission to use Roach to perform backup and restoration.

To create an O&M administrator, connect to the database as an initial user and use the **CREATE USER** or **ALTER USER** statement with the **OPRADMIN** option.

```
gaussdb=# CREATE USER opradmin WITH OPRADMIN password "********";
```

or

```
gaussdb=# ALTER USER joe OPRADMIN;
```

To run the **ALTER USER** statement, the user must exist.

Security Policy Administrator

A security policy administrator is an account with the **POLADMIN** attribute and has the permission to create resource tags, masking policies, and unified audit policies.

To create a security policy administrator, connect to the database as an administrator and use the **CREATE USER** or **ALTER USER** statement with the **POLADMIN** option.

```
gaussdb=# CREATE USER poladmin WITH POLADMIN password "********";
```

or

```
gaussdb=# ALTER USER joe POLADMIN;
```

To run the **ALTER USER** statement, the user must exist.

2.1.3 Separation of Duties

Descriptions in **Default Permission Mechanism** and **Administrators** are about the initial situation after the database system is created. By default, the system administrator with the **SYSADMIN** attribute has the highest permission in the system.

To avoid risks caused by centralized permissions, you can enable separation of duties to assign the system administrator's user creation permission to security administrators and audit management permission to audit administrators.

After separation of duties, the system administrator does not have the **CREATEROLE** attribute (security administrator) or the **AUDITADMIN** attribute (audit administrator). That is, the system administrator can neither create roles or users, nor view or maintain database audit logs. For details about the **CREATEROLE** and **AUDITADMIN** attributes, see **CREATE ROLE**.

Separation of duties does not take effect for an initial user. Therefore, you are advised to use an initial user as a database administrator only for database management other than service running.

To enable separation of duties, set the GUC parameter **enableSeparationOfDuty** to **on**.

WARNING

If you need to use the separation of duties model, specify it during database initialization. You are advised not to switch the permission management model back and forth. In particular, if you want to switch from a non-separation-of-duties permission management model to the separation-of-duties permission management model, you need to review the permission set of existing users. If a user has the system administrator permission and audit administrator permission, the permissions need to be tailored.

After separation of duties, the system administrator does not have permissions for non-system schemas of other users. Therefore, the system administrator cannot access the objects in other users' schemas before being granted the permissions.

For details about permission changes before and after enabling separation of duties, see **Table 2-1** and **Table 2-2**.

Table 2-1 Default user permissions

Object Name	Initial User (ID: 10)	System Administrator	Security Administrator	Audit Administrator	Common User
Tablespaces	Has all permissions.	Can create, modify, delete, access, or grant permissions for tablespaces.	Cannot create, modify, delete, or grant permissions for tablespaces and can access tablespaces if the access permission is granted.		
Schemas		Has all permissions for all schemas except **dbe_perf**.	Has all permissions for their own schemas, but does not have permissions for non-system schemas of other users.		
User-defined functions		Has all permissions for all user-defined functions.	Has all permissions for their own functions, and has only the call permission for other users' functions.		
User-defined tables or views		Has all permissions for all user-defined tables or views.	Has all permissions for their own tables or views, but does not have permissions for other users' tables or views.		

Table 2-2 Changes in permissions after separation of duties

Object Name	Initial User (ID: 10)	System Administrator	Security Administrator	Audit Administrator	Common User
Tablespaces	N/A Has all permissions.	N/A	N/A		
Schemas		Permissions reduced Has all permissions for their own schemas, but does not have permissions for non-system schemas of other users.	N/A		
User-defined functions		Cannot access functions in non-system schemas of other users before being granted the permissions.	N/A		

Object Name	Initial User (ID: 10)	System Administrator	Security Administrator	Audit Administrator	Common User
User-defined tables or views		Cannot access tables or views in non-system schemas of other users before being granted the permissions.	N/A		

NOTICE

PG_STATISTIC and PG_STATISTIC_EXT store sensitive information about statistical objects, such as high-frequency MCVs. After separation of duties is enabled, the system administrator can still access the two system catalogs to obtain sensitive information in the statistics.

2.1.4 Users

You can use CREATE USER and ALTER USER to create and manage database users, respectively. A database system contains one or more databases. Users and roles are shared within the entire database system, but their data is not shared. That is, a user can connect to any database, but after the connection is successful, any user can access only the database declared in the connection request.

In non-**Separation of Duties** scenarios, GaussDB user accounts can be created and deleted only by a system administrator or a security administrator with the **CREATEROLE** attribute. In separation-of-duties scenarios, a user account can be created only by an initial user or a security administrator.

When a user logs in, GaussDB authenticates the user. A user can own databases and database objects (such as tables), and grant permissions of these objects to other users and roles. In addition to system administrators, users with the **CREATEDB** attribute can create databases and grant permissions on these databases.

Adding, Modifying, and Deleting Users

- To create a user, use the SQL statement **CREATE USER**.

  For example, create user **joe** and set the **CREATEDB** attribute for the user.

  ```
  gaussdb=# CREATE USER joe WITH CREATEDB PASSWORD "********";
  CREATE ROLE
  ```

- To create a system administrator, use the **CREATE USER** statement with the **SYSADMIN** option.
- To delete an existing user, use **DROP USER**.
- To change a user account (for example, rename the user or change the password), use **ALTER USER**.

- To view a user list, query the **PG_USER** view.
  ```
  gaussdb=# SELECT * FROM pg_user;
  ```
- To view user attributes, query the **PG_AUTHID** system catalog.
  ```
  gaussdb=# SELECT * FROM pg_authid;
  ```

Permanent User

GaussDB provides a permanent user solution. You can create a permanent user with the **PERSISTENCE** attribute, which can use the service_reserved_connections channel to connect to the database.

```
gaussdb=# CREATE USER user_persistence WITH PERSISTENCE IDENTIFIED BY "********";
```

Only the initial user is allowed to create, modify, and delete permanent users with the **PERSISTENCE** attribute.

2.1.5 Roles

After a role is granted to a user through GRANT, the user will have all the permissions of the role. It is recommended that roles be used to efficiently grant permissions. For example, you can create different roles of design, development, and maintenance personnel, grant the roles to users, and then grant specific data permissions required by different users. When permissions are granted or revoked at the role level, these changes take effect on all members of the role.

GaussDB provides an implicitly defined group **PUBLIC** that contains all roles. By default, all new users and roles have the permissions of **PUBLIC**. For details about the default permissions of **PUBLIC**, see **GRANT**. To revoke permissions of **PUBLIC** from a user or role, or re-grant these permissions to them, add the PUBLIC keyword in the REVOKE or GRANT statement.

To view all roles, query the system catalog PG_ROLES.

```
SELECT * FROM PG_ROLES;
```

Adding, Modifying, and Deleting Roles

In scenarios other than **Separation of Duties**, a role can be created, modified, and deleted only by a system administrator or a user with the **CREATEROLE** attribute. In separation-of-duties scenarios, a role can be created, modified, and deleted only by an initial user or a user with the **CREATEROLE** attribute.

- To create a role, use **CREATE ROLE**.
- To add or delete users in an existing role, use **ALTER ROLE**.
- To delete a role, use **DROP ROLE**. DROP ROLE deletes only a role, rather than member users in the role.

Built-in Roles

GaussDB provides a group of default roles whose names start with **gs_role_**. These roles are provided to access to specific, typically high-privileged operations. You can grant these roles to other users or roles within the database so that they can use specific functions. These roles should be given with great care to ensure that they are used where they are needed. **Table 2-3** describes the permissions of built-in roles.

Table 2-3 Permission description of built-in roles

Roles	Permission
gs_role_signal_backend	Permission to call the pg_cancel_backend, pg_terminate_backend, and pg_terminate_session functions to cancel or terminate other sessions. However, this role cannot perform operations on sessions of the initial user or **PERSISTENCE** user.
gs_role_tablespace	Permission to create a tablespace.
gs_role_replication	Permission to call logical replication functions, such as **kill_snapshot**, **pg_create_logical_replication_slot**, **pg_create_physical_replication_slot**, **pg_drop_replication_slot**, **pg_replication_slot_advance**, **pg_create_physical_replication_slot_extern**, **pg_logical_slot_get_changes**, **pg_logical_slot_peek_changes**, **pg_logical_slot_get_binary_changes**, and **pg_logical_slot_peek_binary_changes**.
gs_role_account_lock	Permission to lock and unlock users. However, this role cannot lock or unlock the initial user or users with the **PERSISTENCE** attribute.
gs_role_pldebugger	Permission to debug functions in **dbe_pldebugger**.
gs_role_public_dblink_drop	Permission to delete public database links.
gs_role_public_dblink_alter	Permission to modify public database links.
gs_role_seclabel	Permission to create, delete, and apply security labels.
gs_role_public_synonym_create	Permission to create public synonyms.
gs_role_public_synonym_drop	Permission to delete public synonyms.

The restrictions on built-in roles are as follows:

- The role names starting with **gs_role_** are reserved for built-in roles in the database. Do not create users, roles, or schemas starting with **gs_role_** or rename existing users, roles, or schemas to names starting with **gs_role_**.
- Do not perform ALTER or DROP operations on built-in roles.
- By default, built-in roles do not have the LOGIN permission and do not have preset passwords.
- The gsql meta-commands **\du** and **\dg** do not display information about built-in roles. However, if *pattern* is set to a specific built-in role, the information is displayed.

- When separation of duties is disabled, the initial user, users with the SYSADMIN permission, and users with the ADMIN OPTION built-in role permission have the permission to perform GRANT and REVOKE operations on built-in roles. When separation of duty is enabled, the initial user and users with the ADMIN OPTION built-in role permission have the permission to perform GRANT and REVOKE operations on built-in roles. For example:

```
GRANT gs_role_signal_backend TO user1;
REVOKE gs_role_signal_backend FROM user1;
```

2.1.6 Schemas

Schemas function as models. Schemas allow multiple users to use the same database without interference. In this way, database objects can be organized into logical groups that are easy to manage, and third-party applications can be added to corresponding schemas without causing conflicts.

Each database has one or more schemas. Each schema contains tables and other types of objects. When a database is created, a public schema named **public** is created by default, and all users have the USAGE permission on this schema. In addition, each database has a pg_catalog schema, which contains system catalogs and all built-in data types, functions, and operators. Only system administrators and initial users can create common functions, aggregate functions, stored procedures, and synonym objects in public and pg_catalog schemas. Only initial users can create operators in public and pg_catalog schemas. Other users cannot create the preceding five types of objects even if they are granted the CREATE permission on the public and pg_catalog schemas. You can group database objects by schema. A schema is similar to an OS directory but cannot be nested. By default, only the initial user can create objects in pg_catalog.

The same database object name can be used in different schemas of the same database without causing conflicts. For example, both a_schema and b_schema can contain a table named **mytable**. Users with required permissions can access objects across multiple schemas of the same database.

When you run the **CREATE USER** command to create a user, the system creates a schema with the same name as the user in the database where the command is executed.

Database objects are generally created in the first schema in a database search path. For details about the first schema and how to change the schema order, see **Search Path**.

Creating, Modifying, and Deleting Schemas

- Create a schema. For details, see **CREATE SCHEMA**. By default, the initial user and system administrator can create schemas. Other users can create schemas in the database only when they have the CREATE permission on the database. For details about how to grant the permission, see the syntax in **GRANT**.
- To change the name or owner of a schema, use **ALTER SCHEMA**. The schema owner can change a schema.
- To delete a schema and its objects, use **DROP SCHEMA**. The schema owner can delete a schema.

- To create a table in a schema, use the *schema_name*.*table_name* format to specify the table. If *schema_name* is not specified, the table will be created in the first schema in **Search Path**.
- To view the owner of a schema, perform the following join query on the system catalogs PG_NAMESPACE and PG_USER. Replace *schema_name* in the statement with the name of the schema to be queried.
  ```
  gaussdb=# SELECT s.nspname,u.usename AS nspowner FROM pg_namespace s, pg_user u WHERE nspname='schema_name' AND s.nspowner = u.usesysid;
  ```
- To view a list of all schemas, query the system catalog PG_NAMESPACE.
  ```
  gaussdb=# SELECT * FROM pg_namespace;
  ```
- To view a list of tables in a schema, query the system catalog PG_TABLES. For example, the following query will return a table list from PG_CATALOG in the schema.
  ```
  gaussdb=# SELECT distinct(tablename),schemaname from pg_tables where schemaname = 'pg_catalog';
  ```

Search Path

A search path is defined in the **search_path** parameter. The parameter value is a list of schema names separated by commas (,). If no target schema is specified during object creation, the object will be added to the first schema listed in the search path. If there are objects with the same name across different schemas and no schema is specified for an object query, the object will be returned from the first schema containing the object in the search path.

- To view the current search path, use **SHOW**.
  ```
  gaussdb=# SHOW SEARCH_PATH;
   search_path
  ----------------
   "$user",public
  (1 row)
  ```
  The default value of **search_path** is **"***$user***",public**. *$user* indicates the name of the schema with the same name as the current session user. If the schema does not exist, *$user* will be ignored. By default, after a user connects to a database that has schemas with the same name, objects will be added to all the schemas. If there are no such schemas, objects will be added only to the public schema.
- To change the default schema of the current session, run the SET statement.

  Set the search path to **myschema, public** (**myschema** will be searched first).
  ```
  gaussdb=# SET SEARCH_PATH TO myschema, public;
  SET
  ```

2.1.7 User Permissions

- To grant permissions for an object to a user, use **GRANT**.

  When permissions for a table or view in a schema are granted to a user or role, the USAGE permission of the schema must be granted together. Otherwise, the user or role can only see these objects but cannot access them.

  In the following example, permissions for the schema tpcds are first granted to user **joe**, and then the SELECT permission for the tpcds.web_returns table is also granted.
  ```
  gaussdb=# GRANT USAGE ON SCHEMA tpcds TO joe;
  gaussdb=# GRANT SELECT ON TABLE tpcds.web_returns to joe;
  ```


- Grant a role to a user to allow the user to inherit the object permissions of the role.

  a. Create a role.

     Create a role **lily** and grant the system permission CREATEDB to the role.

     ```
     gaussdb=# CREATE ROLE lily WITH CREATEDB PASSWORD "********";
     ```

  b. For details about how to grant object permissions to a role, see **GRANT**.

     For example, first grant permissions for the schema tpcds to the role **lily**, and then grant the SELECT permission of the tpcds.web_returns table to **lily**.

     ```
     gaussdb=# GRANT USAGE ON SCHEMA tpcds TO lily;
     gaussdb=# GRANT SELECT ON TABLE tpcds.web_returns to lily;
     ```

  c. Grant the role permissions to a user.

     ```
     gaussdb=# GRANT lily to joe;
     ```

     NOTE

     When the permissions of a role are granted to a user, the attributes of the role are not transferred together.

- For details about how to revoke user permissions, see **REVOKE**.

2.1.8 Row-Level Security Policy

The row-level security feature enables database access control to be accurate to each row of data tables. In this way, the same SQL query may return different results for different users.

You can create a row-level security policy for a data table. The policy defines an expression that takes effect only for specific database users and SQL operations. When a database user accesses the data table, if an SQL statement meets the specified row-level security policies of the data table, the expressions that meet the specified condition will be combined by using AND or OR based on the attribute type (PERMISSIVE | RESTRICTIVE) and applied to the execution plan in the query optimization phase.

Row-level security policy is used to control the visibility of row-level data in tables. By predefining filters for data tables, the expressions that meet the specified condition can be applied to execution plans in the query optimization phase, which will affect the final execution result. Currently, the SQL statements that can be affected include SELECT, UPDATE, and DELETE.

Scenario 1: A table summarizes the data of different users. Users can view only their own data.

```
-- Create users alice, bob, and peter.
gaussdb=# CREATE USER alice PASSWORD '********';
gaussdb=# CREATE USER bob PASSWORD '********';
gaussdb=# CREATE USER peter PASSWORD '********';

-- Create the all_data table that contains user information.
gaussdb=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));

-- Insert data into the data table.
gaussdb=# INSERT INTO all_data VALUES(1, 'alice', 'alice data');
gaussdb=# INSERT INTO all_data VALUES(2, 'bob', 'bob data');
gaussdb=# INSERT INTO all_data VALUES(3, 'peter', 'peter data');

-- Grant the read permission for the all_data table to users alice, bob, and peter.
```



```
gaussdb=# GRANT SELECT ON all_data TO alice, bob, peter;

-- Enable the row-level security policy.
gaussdb=# ALTER TABLE all_data ENABLE ROW LEVEL SECURITY;

-- Create a row-level security policy to specify that the current user can view only their own data.
gaussdb=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role = CURRENT_USER);

-- View table details.
gaussdb=# \d+ all_data
                            Table "public.all_data"
 Column |          Type          | Modifiers | Storage  | Stats target | Description
--------+------------------------+-----------+----------+--------------+-------------
 id     | integer                |           | plain    |              |
 role   | character varying(100) |           | extended |              |
 data   | character varying(100) |           | extended |              |
Row Level Security Policies:
    POLICY "all_data_rls" FOR ALL
      TO public
      USING (((role)::name = "current_user"()))
Has OIDs: no
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, enable_rowsecurity=true

-- Switch to user alice and run SELECT * FROM public.all_data.
gaussdb=# SELECT * FROM public.all_data;
 id | role  |    data
----+-------+------------
  1 | alice | alice data
(1 row)

gaussdb=# EXPLAIN(COSTS OFF) SELECT * FROM public.all_data;
                        QUERY PLAN
-----------------------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on all_data
         Filter: ((role)::name = 'alice'::name)
 Notice: This query is influenced by row level security feature
(5 rows)

-- Switch to user peter and run SELECT * FROM public.all_data.
gaussdb=# SELECT * FROM public.all_data;
 id | role  |    data
----+-------+------------
  3 | peter | peter data
(1 row)

gaussdb=#  EXPLAIN(COSTS OFF) SELECT * FROM public.all_data;
                        QUERY PLAN
-----------------------------------------------------------------
 Streaming (type: GATHER)
   Node/s: All datanodes
   ->  Seq Scan on all_data
         Filter: ((role)::name = 'peter'::name)
 Notice: This query is influenced by row level security feature
(5 rows)
```

NOTICE

PG_STATISTIC and PG_STATISTIC_EXT store sensitive information about statistical objects, such as high-frequency MCVs. If the permission to query the two system catalogs is granted to a common user after the row-level security policy is created, the common user can still access the two system catalogs to obtain sensitive information in the statistical objects.

2.2 Database Audit

Context

Database audit is critical to the security of the database system. Database audit administrators can use the audit logs to reproduce a series of events that cause faults in the database and identify unauthorized users, unauthorized operations, and the time when these operations are performed.

You need to understand the following information about the audit function:

- The overall audit switch **audit_enabled** supports dynamic loading. After you change the switch status when the database is running, the change takes effect immediately and you do not need to restart the database. Its default value is **on**, indicating that the audit function is enabled.
- In addition to the overall audit switch, each audit item has an independent switch. An audit item is available only after its own switch is turned on.
- The switch of each audit supports dynamic loading. After you change the audit switch when the database is running, the change takes effect immediately and you do not need to restart the database.

Table 2-4 describes the audit items supported by GaussDB. If you need to modify a specific audit configuration item, contact the administrator. For details about parameter types and value ranges, see "Configuring Running Parameters > GUC Parameters > Auditing" in *Administrator Guide*.

Table 2-4 Audit items

Configuration Item	Description
User login and logout audit	Parameter: **audit_login_logout** Its default value is **7**, which indicates that the function of user login and logout audit is enabled. **0** indicates that the function of user login and logout audit is disabled. Other values are not recommended.
Database startup, stop, recovery, and switchover audit	Parameter: **audit_database_process** Its default value is **1**, which indicates that the audit of database startup, stop, recovery, and switchover is enabled.
User locking and unlocking audit	Parameter: **audit_user_locked** Its default value is **1**, which indicates that the audit of user locking and unlocking is enabled.
Unauthorized access audit	Parameter: **audit_user_violation** Its default value is **0**, which indicates that the audit of unauthorized access disabled.

Configuration Item	Description
Permission granting and revoking audit	Parameter: **audit_grant_revoke** Its default value is **1**, which indicates that the audit of permission granting and revoking is enabled.
Full audit of user operations	Parameter: **full_audit_users** Its default value is an empty string, indicating that the default configuration is used and no full audit of user operations is configured.
Names and IP addresses of clients that do not need to be audited	Parameter: **no_audit_client** Its default value is an empty string, indicating that the default configuration is used and no clients and IP addresses are added to the audit blacklist.
Audit of CREATE, ALTER, and DROP operations on database objects	Parameter: **audit_system_object** Its default value is **67121159**, which indicates that the CREATE, ALTER, and DROP operations only on databases, schemas, users, and SQL patches are audited.
Audit of INSERT, UPDATE, and DELETE operations on a specific table	Parameter: **audit_dml_state** Its default value is **0**, which indicates that the audit of DML operations (except SELECT) on a specific table is disabled.
SELECT operation audit	Parameter: **audit_dml_state_select** Its default value is **0**, which indicates that the audit of the SELECT operation is disabled.
COPY operation audit	Parameter: **audit_copy_exec** Its default value is **1**, which indicates that the audit of COPY operations is enabled.
Audit of execution of stored procedures and customized functions	Parameter: **audit_function_exec** Its default value is **0**, which indicates that no execution audit logs of stored procedures and customized functions are recorded.
Audit of system functions in the whitelist	Parameter: **audit_system_function_exec** Its default value is **0**, which indicates that audit logs of system function execution are not recorded.
SET operation audit	Parameter: **audit_set_parameter** Its default value is **0**, which indicates that the audit of the SET operation is disabled.
Transaction ID record	Parameter: **audit_xid_info** Its default value is **0**, which indicates that the function of recording transaction IDs in audit logs is disabled.

Configuration Item	Description
Audit of internal tool connections and operations	Parameter: **audit_internal_event** The default value is **off**, which indicates that the login, logout, and other SQL operations of the internal tools cm_agent, gs_clean, and WDRXdb are not audited.

3 Database Quick Start

3.1 Operating a Database

This section explains how to use databases, including creating database accounts, databases, and tables, inserting data to tables, and querying data in tables.

3.1.1 Creating a Database Account

Only administrators that are created during database installation can access the initial database by default. You can also create other database accounts.

```
gaussdb=# CREATE USER joe WITH PASSWORD "********";
```

If the following information is displayed, the creation is successful:

```
CREATE ROLE
```

In this case, you have created a user account named **joe**, and the user password is **********.

Set user **joe** as the system administrator.

```
gaussdb=# GRANT ALL PRIVILEGES TO joe;
```

Run the **GRANT** command to set related permissions. For details, see **GRANT**.

NOTE

For details about how to create database accounts, see **Users and Permissions**.

3.1.2 Creating and Managing Databases

Prerequisites

Only database system administrators or users granted with database creation permissions can create a database. For details about how to grant database creation permissions to a user, see **Users and Permissions**.

Context

- GaussDB has three default template databases **template0**, **template1**, and **templatem**, and a default user database **postgres**. The default compatible database type of Postgres is O (that is, **DBCOMPATIBILITY** is set to **A**). In this compatible type, empty strings are processed as null values.
- CREATE DATABASE creates a database by copying a template database (**template0** by default). Do not use a client or other methods to connect to or operate the two template databases.

  NOTE

  - The template database does not contain any user table. You can view the attributes of the template database in the **PG_DATABASE** system catalog.
  - The **template0** template does not allow user connections. Only initial users of the database and system administrators can connect to **template1** and **templatem**.
- A database system consists of multiple databases. A client can connect to only one database at a time. Users cannot query data across databases. If GaussDB contains multiple databases, set the **-d** parameter to specify the database instance to be connected.

Precautions

Assume that the database encoding is SQL_ASCII. (You can run the **show server_encoding;** command to query the encoding used for storing data in the current database.) If the database object name contains multi-byte characters (such as Chinese) or if the object name length exceeds the allowed maximum (63 bytes), the database truncates the last byte (not the last character) of the object name. In this case, half characters may appear.

To resolve this problem, you need to:

- Ensure that the name of the data object does not exceed the maximum length.
- Change the default database storage code set (**server_encoding**) to UTF-8.
- Exclude multi-byte characters from object names.
- If you fail to delete an object by specifying its name after truncation, specify its original name to delete it, or manually delete it from the system catalogs on each node.

Procedure

Step 1 Create a database named **db_tpcc**.

```
gaussdb=# CREATE DATABASE db_tpcc;
CREATE DATABASE
```

NOTE

- Database names must comply with the general naming convention rules of SQL identifiers. The current role automatically becomes the owner of this new database.
- If a database system is used to support independent users and projects, store them in different databases.
- If the projects or users are associated with each other and share resources, store them in one database. However, you can divide them into different schemas. A schema is a logical structure, and the access permission for a schema is controlled by the permission system module.
- A database name contains a maximum of 63 bytes and the excessive bytes at the end of the name will be truncated by the server. You are advised to specify a database name no longer than 63 bytes when you create a database.
- New databases are created in the **pg_default** tablespace by default. To specify another tablespace, run the following statement:

```
gaussdb=# CREATE DATABASE db_tpcc WITH TABLESPACE = hr_local;
CREATE DATABASE
```

  hr_local indicates the tablespace name. For details about how to create a tablespace, see **Creating and Managing Tablespaces**.
- After creating the **db_tpcc** database, you can perform other operations in the default **postgres** database. Alternatively, you can perform the following operations to exit the **postgres** database, connect to the **db_tpcc** database as a new user, and perform operations such as creating tables:

```
gaussdb=# \q
gsql -d db_tpcc -p 8000 -U joe
Password for user joe:
gsql ((GaussDB Kernel XXX.X.XXX build f521c606) compiled at 2021-09-16 14:55:22 commit 2935 last mr 6385 release)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

db_tpcc=>
```

Step 2 View databases.

- Run the **\l** meta-command to view the database list of the database system.

```
gaussdb=# \l
```

- Query the database list in the pg_database system catalog.

```
gaussdb=# SELECT datname FROM pg_database;
```

Step 3 Modify the database.

You can modify database configuration such as the database owner, name, and default settings.

- Set the default search path for the database.

```
gaussdb=# ALTER DATABASE db_tpcc SET search_path TO pa_catalog,public;
ALTER DATABASE
```

- Rename the database.

```
gaussdb=# ALTER DATABASE db_tpcc RENAME TO human_tpcds;
ALTER DATABASE
```

Step 4 Delete the database.

You can run the **DROP DATABASE** command to delete a database. This command deletes the system catalog of the database and the database directory on the disk. Only the database owner or system administrator can delete a database. A database accessed by users cannot be deleted. You need to connect to another database before deleting this database.

Delete the database.


```
gaussdb=# DROP DATABASE human_tpcds;
DROP DATABASE
```

----End

3.1.3 Creating and Managing Tablespaces

Context

The administrator can use tablespaces to control the layout of disks where a database is installed. This has the following advantages:

- If the disk partition or tablespace initially allocated to the database is full and the space cannot be logically extended, you can create and use tablespaces in other partitions until the system space is reconfigured.
- Tablespaces allow the administrator to distribute data based on the schema of database objects, improving system performance.
  - A frequently used index can be stored in a disk having stable performance and high computing speed, such as a solid device.
  - A table that stores archived data and is rarely used or has low performance requirements can be placed in a disk with a slow computing speed.
- You can use tablespaces to limit the disk space of databases. If a tablespace shares a partition with other tablespaces, the tablespace will never occupy the space allocated to other tablespaces.
- You can use tablespaces to manage the disk space used to store database data. The database will switch to the read-only mode when the disk usage of the tablespace reaches 90%. Once the disk usage drops below 90%, the database will be restored to the read-write mode.

  You are advised to use the background monitoring program or Database Manager to monitor the disk space usage when using the database to prevent the database from switching to the read-only mode.
- Each tablespace corresponds to a file system directory. If *data directory* **/ pg_location/mount1/path1** is an empty directory for which users have read and write permissions, an absolute path tablespace can be created in this directory.

  If the tablespace quota management is used, the performance may deteriorate by about 30%. **MAXSIZE** specifies the maximum quota for each database node. The deviation must be within 500 MB. Determine whether to set a tablespace to its maximum size as required.

GaussDB provides two tablespaces: pg_default and pg_global.

- Default tablespace pg_default: stores non-shared system catalogs, user tables, user table indexes, temporary tables, temporary table indexes, and internal temporary tables. The corresponding storage directory is the base directory in the instance data directory.
- Shared tablespace pg_global: stores shared system catalogs. The corresponding storage directory is the base directory in the global data directory.

Precautions

You are advised not to use user-defined tablespaces in the Huawei Cloud scenario. This is because user-defined tablespaces are usually used with storage media other than the main storage (storage device where the default tablespace is located, such as a disk) to isolate I/O resources that can be used by different services. Storage devices use standard configurations and do not have other available storage media in the Huawei Cloud scenario. If the user-defined tablespace is not properly used, the system cannot run stably for a long time and the overall performance is affected. Therefore, you are advised to use the default tablespace.

Procedure

- Create a tablespace.
  a. Create user **jack** and set the password to **********.
  ```
  gaussdb=# CREATE USER jack IDENTIFIED BY '********';
  ```
  If the following information is displayed, the creation is successful:
  ```
  CREATE ROLE
  ```
  b. Create a tablespace.
  ```
  gaussdb=# CREATE TABLESPACE fastspace RELATIVE LOCATION 'tablespace/tablespace_1';
  ```
  If the following information is displayed, the creation is successful:
  ```
  CREATE TABLESPACE
  ```
  fastspace is the new tablespace, and *Database node data directory*/**pg_location/tablespace/tablespace_1** is an empty directory for which users have read and write permissions.
  c. A database system administrator can run the following command to grant the permission of accessing the **fastspace** tablespace to user **jack**:
  ```
  gaussdb=# GRANT CREATE ON TABLESPACE fastspace TO jack;
  ```
  If the following information is displayed, the grant operation is successful:
  ```
  GRANT
  ```
- Create an object in a tablespace.
  If you have the CREATE permission on the tablespace, you can create database objects in the tablespace, such as tables and indexes.
  Take creating a table as an example:
  – Method 1: Create a table in a specified tablespace.
  ```
  gaussdb=# CREATE TABLE foo(i int) TABLESPACE fastspace;
  ```
  If the following information is displayed, the creation is successful:
  ```
  CREATE TABLE
  ```
  – Method 2: Run **SET default_tablespace** to set the default tablespace and then create a table.
  ```
  gaussdb=# SET default_tablespace = 'fastspace';
  SET
  gaussdb=# CREATE TABLE foo2(i int);
  CREATE TABLE
  ```
  In this example, **fastspace** is the default tablespace, and **foo2** is the created table.
- Use one of the following methods to query a tablespace:
  – Method 1: Check the pg_tablespace system catalog. View all the tablespaces defined by the system and users.


```
gaussdb=# SELECT spcname FROM pg_tablespace;
```

  - Method 2: Run the following meta-command of the gsql program to query the tablespaces:

```
gaussdb=# \db
```

- Query the tablespace usage.
  a. Query the current usage of the tablespace.

```
gaussdb=# SELECT PG_TABLESPACE_SIZE('fastspace');
```

     Information similar to the following is displayed:

```
 pg_tablespace_size
--------------------
            2146304
(1 row)
```

     2146304 is the size of the tablespace, and its unit is byte.

  b. Calculate the tablespace usage.

     Tablespace usage = Value of **PG_TABLESPACE_SIZE**/Size of the disk where the tablespace resides

- Modify a tablespace.

  Rename tablespace **fastspace** to **fspace**.

```
gaussdb=# ALTER TABLESPACE fastspace RENAME TO fspace;
ALTER TABLESPACE
```

- Delete a tablespace and related data.
  - Delete user **jack**.

```
gaussdb=# DROP USER jack CASCADE;
DROP ROLE
```

  - Drop tables **foo** and **foo2**.

```
gaussdb=# DROP TABLE foo;
gaussdb=# DROP TABLE foo2;
```

    If the following information is displayed, the operation is successful:

```
DROP TABLE
```

  - Drop tablespace **fspace**.

```
gaussdb=# DROP TABLESPACE fspace;
DROP TABLESPACE
```

    NOTE

    Only the tablespace owner or system administrator can drop a tablespace.

3.1.4 Creating and Managing Tables

3.1.4.1 Creating Tables

Context

A table is created in a database and can be saved in different databases. Tables under different schemas in a database can have the same name.

Procedure

Create a table.

```
gaussdb=# CREATE TABLE customer_t1
(
    c_customer_sk             integer,
```

```
    c_customer_id         char(5),
    c_first_name          char(6),
    c_last_name           char(8)
);
```

If the following information is displayed, the creation is successful:

```
CREATE TABLE
```

c_customer_sk, **c_customer_id**, **c_first_name**, and **c_last_name** are the column names of the table. **integer**, **char(5)**, **char(6)**, and **char(8)** are column name types.

NOTE

- By default, new database objects are created in the $user schema. For more details about schemas, see **Creating and Managing Schemas**.
- In addition to the created tables, a database contains many system catalogs. These system catalogs contain database installation information and information about various queries and processes in GaussDB. You can collect information about the database by querying system catalogs. For details, see **Querying System Catalogs**.
- For more details about how to create a table, see **CREATE TABLE**.

3.1.4.2 Inserting Data to Tables

A new table contains no data. You need to insert data to the table before using it. This section describes how to insert a row or multiple rows of data using the **INSERT** command and to insert data from a specified table. Contact the administrator if a large amount of data needs to be imported.

Context

The length of a character on the server and client may vary by the character sets they use. A string entered on the client will be processed based on the server's character set, so the result may differ from expected.

Table 3-1 Comparison of character set output between the client and server

Procedure	Server and Client Use Same Encoding	Server and Client Use Different Encoding
No operations are performed on the string while it is saved and read.	Your expected result is returned.	If the encoding for input and output on the client is the same, your expected result is returned.
Operations (such as executing string functions) are performed to the string while it is saved and read.	Your expected result is returned.	The result may not be as expected, depending on the operations performed on the string.

Procedure	Server and Client Use Same Encoding	Server and Client Use Different Encoding
A long string is truncated while it is saved.	Your expected result is returned.	If the character sets used on the client and server have different character length, the result may not be as expected.

More than one of the preceding operations can be performed to a string. For example, if the character sets of the client and server are different, a string may be processed and then truncated. In this case, the result will also be unexpected. For details, see **Table 3-2**.

NOTE

Long strings are truncated only if **DBCOMPATIBILITY** is set to **TD** and the GUC parameter **td_compatible_truncation** is set to **on**.

Create **table1** and **table2** to be used in the example.

```
gaussdb=# CREATE TABLE table1(id int, a char(6), b varchar(6),c varchar(6));
gaussdb=# CREATE TABLE table2(id int, a char(20), b varchar(20),c varchar(20));
```

Table 3-2 Examples

No.	Server Character Set	Client Character Set	Automatic Truncation Enabled	Example	Result	Description
1	SQL_ASCII	UTF8	Yes	gaussdb=# **INSERT INTO** *table1* **VALUES**(*1,reverse('123AA78'),reverse('123AA78'),reverse('123AA78')*);	id |a|b|c ----+------ +------+------ 1 | 87| 87| 87	A string is reversed on the server and then truncated. Because character sets used by the server and client are different, character A is displayed in multiple bytes on the server and the result is incorrect.

No.	Server Character Set	Client Character Set	Automatic Truncation Enabled	Example	Result	Description
2	SQL_ASCII	UTF8	Yes	gaussdb=# **INSERT INTO** *table1* **VALUES**(*2,reverse('123A78'),reverse('123A78'),reverse('123A78')*);	id |a|b|c ----+------+------+------ 2 | 873| 873| 873	A string is reversed and then automatically truncated. Therefore, the result is unexpected.
3	SQL_ASCII	UTF8	Yes	gaussdb=# **INSERT INTO** *table1* **VALUES**(*3,'87A123','87A123','87A123'*);	id | a | b | c ----+-------+-------+------- 3 | 87A1 | 87A1 | 87A1	The length of the column of the string type is an integer multiple of the length in client character encoding. Therefore, the result is correct after truncation.
4	SQL_ASCII	UTF8	No	gaussdb=# **INSERT INTO** *table2* **VALUES**(*1,reverse('123AA78'),reverse('123AA78'),reverse('123AA78')*); gaussdb=# **INSERT INTO** *table2* **VALUES**(*2,reverse('123A78'),reverse('123A78'),reverse('123A78')*);	id |a|b|c ---- +---------- --+-------- +-------- 1 | 87 321| 87 321 | 87 321 2 | 87321| 87321| 87321	Similar to the first example, multi-byte characters no longer indicate the original characters after being reversed.

Procedure

You need to create a table before inserting data to it. For details about how to create a table, see **Creating and Managing Tables**.

- Insert a row to table **customer_t1**.

  Data values are arranged in the same order as the columns in the table and are separated by commas (,). Generally, column values are text values (constants). But column values can also be scalar expressions.

  ```
  gaussdb=# INSERT INTO customer_t1(c_customer_sk, c_customer_id, c_first_name) VALUES (3769, 'hello', 'Grace');
  ```


If you know the sequence of the columns in the table, you can obtain the same result without listing these columns. For example, the following command generates the same result as the preceding command:

```
gaussdb=# INSERT INTO customer_t1 VALUES (3769, 'hello', 'Grace');
```

If you do not know some of the column values, you can omit them. In the INSERT statement, if the column name of the specified target table is not displayed, the values to be inserted in the VALUES clause correspond to the columns of the target table based on the column number. That is, the first value in the VALUES clause corresponds to the first column of the target table, the second value of the VALUES clause corresponds to the second column of the target table, and so on. Columns that do not have corresponding values in the VALUES clause are automatically filled with default values or NULL. For example:

```
gaussdb=# INSERT INTO customer_t1 (c_customer_sk, c_first_name) VALUES (3769, 'Grace');

gaussdb=# INSERT INTO customer_t1 VALUES (3769, 'hello');
```

You can also specify the default value of a column or row.

```
gaussdb=# INSERT INTO customer_t1 (c_customer_sk, c_customer_id, c_first_name) VALUES (3769, 'hello', DEFAULT);

gaussdb=# INSERT INTO customer_t1 DEFAULT VALUES;
```

- To insert multiple rows to a table, run the following command:

```
gaussdb=# INSERT INTO customer_t1 (c_customer_sk, c_customer_id, c_first_name) VALUES
    (6885, 'maps', 'Joes'),
    (4321, 'tpcds', 'Lily'),
    (9527, 'world', 'James');
```

  You can also insert multiple rows by running the command for inserting one row for multiple times. However, you are advised to run this command to improve efficiency.

- Assume that you have created a backup table **customer_t2** for table **customer_t1**. To insert data from **customer_t1** to **customer_t2**, run the following commands:

```
gaussdb=# CREATE TABLE customer_t2
(
    c_customer_sk            integer,
    c_customer_id            char(5),
    c_first_name             char(6),
    c_last_name              char(8)
);

gaussdb=# INSERT INTO customer_t2 SELECT * FROM customer_t1;
```

  NOTE

  If implicit conversion is not implemented between the column data types of the specified table and those of the current table, the two tables must have the same column data types when data is inserted from the specified table to the current table.

- Insert data into a table by using a table alias without the keyword AS.

```
gaussdb=# CREATE TABLE T1(A INT,B TEXT,C TIMESTAMP,D FLOAT);
gaussdb=# INSERT INTO T1 T VALUES(1,'HA','1997-01-01 10:00:00'::TIMESTAMP,1.234);
gaussdb=# INSERT INTO T1 TT(B,C,D) VALUES('HA','1997-01-01 10:00:00'::TIMESTAMP,1.234);
```


NOTE

- For the INSERT statement, the table alias without the keyword AS cannot be a keyword for statements (such as SELECT or VALUE) or expressions. The alias must comply with the identifier naming rule.
- When the INSERT statement is used without AS, the table alias does not support this format: INSERT INTO table_name alias_name(alias_name.col1,...,alias_name.coln) VALUES(xxx).
- To insert data into a partitioned table by using a table alias without the keyword AS, the target partition cannot be specified.

- To drop a backup table, run the following command:

```
gaussdb=# DROP TABLE customer_t2 CASCADE;
```

NOTE

If the table to be dropped is dependent on other tables, you need to drop its dependent tables first.

3.1.4.3 Updating Data in a Table

Existing data in a database can be updated. You can update one row, all rows, or specified rows of data, or update data in a single column without affecting the data in the other columns.

The following information is required when the UPDATE statement is used to update rows:

- Name of a table and name of a column to be updated
- New column value
- Rows to be updated

Generally, the SQL language does not provide a unique ID for a row of data. Therefore, it is impossible to directly specify the rows of the data to be updated. However, you can specify the rows by declaring the conditions that must be met by the updated row. If a table contains primary keys, you can specify a row using the primary keys.

For details about how to create a table and insert data to it, see **Creating Tables** and **Inserting Data to Tables**.

The value of **c_customer_sk** in the table **customer_t1** must be changed from **9527** to **9876**.

```
gaussdb=# UPDATE customer_t1 SET c_customer_sk = 9876 WHERE c_customer_sk = 9527;
```

You can use a schema to modify the table name. If no such modifier is specified, the table is located based on the default schema path. SET is followed by the column and the new column value. The new value can be a constant or an expression.

For example, increase all the values in the **c_customer_sk** column by 100.

```
gaussdb=# UPDATE customer_t1 SET c_customer_sk = c_customer_sk + 100;
```

This statement does not contain the WHERE clause, so all rows are updated. If the statement contains the WHERE clause, only the rows matching the clause are updated.

In the SET clause, the equal sign (=) indicates value setting. In the WHERE clause, the equal sign indicates comparison. A WHERE condition does not have to be an equality comparison, but can be another operator.

You can run an UPDATE statement to update multiple columns by specifying multiple values in the SET clause. For example:

```
gaussdb=# UPDATE customer_t1 SET  c_customer_id = 'Admin', c_first_name = 'Local' WHERE c_customer_sk = 4421;
```

After data has been updated or deleted in batches, a large number of deletion markers are generated in the data file. During query, data with these deletion markers needs to be scanned as well. In this case, a large amount of data with deletion marks can greatly affect the query performance after batch updates or deletions. If data needs to be updated or deleted in batches frequently, you are advised to periodically run the VACUUM FULL statement to ensure the query performance.

3.1.4.4 Viewing Data

- Run the following command to query information about all tables in a database in the system catalog **pg_tables**:
  ```
  gaussdb=# SELECT * FROM pg_tables;
  ```
- Run the **\d+** command of the **gsql** tool to query table attributes:
  ```
  gaussdb=# \d+ customer_t1;
  ```
- Run the following command to query the data volume of table **customer_t1**:
  ```
  gaussdb=# SELECT count(*) FROM customer_t1;
  ```
- Run the following command to query all data in the table **customer_t1**:
  ```
  gaussdb=# SELECT * FROM customer_t1;
  ```
- Run the following command to query only the data in the column **c_customer_sk**:
  ```
  gaussdb=# SELECT c_customer_sk FROM customer_t1;
  ```
- Run the following command to filter repeated data in the column **c_customer_sk**:
  ```
  gaussdb=# SELECT DISTINCT( c_customer_sk ) FROM customer_t1;
  ```
- Run the following command to query all data whose column **c_customer_sk** is **3869**:
  ```
  gaussdb=# SELECT * FROM customer_t1 WHERE c_customer_sk = 3869;
  ```
- Run the following command to collate data based on the column **c_customer_sk**:
  ```
  gaussdb=# SELECT * FROM customer_t1 ORDER BY c_customer_sk;
  ```

3.1.4.5 Deleting Data from a Table

Outdated data may need to be deleted when tables are used. Data can be deleted from tables only by row.

SQL statements can only access and delete an independent row by declaring conditions that match the row. If a table has a primary key, you can use it to specify a row. You can delete several rows that match the specified condition or delete all the rows from a table.

For example, delete all the rows whose **c_customer_sk** column is **3869** from the table **customer_t1**.

```
gaussdb=# DELETE FROM customer_t1 WHERE c_customer_sk = 3869;
```


To delete all rows from the table, run either of the following commands:

```
gaussdb=# DELETE FROM customer_t1;
```

Or

```
gaussdb=# TRUNCATE TABLE customer_t1;
```

NOTE

If you need to delete an entire table, you are advised to use the **TRUNCATE** statement rather than **DELETE**.

To delete a table, run the following command:

```
gaussdb=# DROP TABLE customer_t1;
```

3.1.5 Querying System Catalogs

In addition to the created tables, a database contains many system catalogs. These system catalogs contain database installation information and information about various queries and processes in GaussDB. You can obtain information about the database by querying system catalogs.

In **System Catalogs and System Views**, the description about each table specifies whether the table is visible to all users or only the initial user. To query tables that are visible only to the initial user, log in as the initial user.

GaussDB provides the following types of system catalogs and views:

- PG-compatible system catalogs and views

  These system catalogs and views have the prefix **PG**.
- New system catalogs and system views of GaussDB

  These system catalogs and views have the prefix **GS**.
- A-compatible system catalogs and views

  These system catalogs and views have the prefix **ALL**, **DBA**, **USER**, or **PV**.

Querying Database Tables

Create the following tables in the public schema:

```
gaussdb=#  CREATE TABLE public.search_table_t1(a int);
CREATE TABLE
gaussdb=#  CREATE TABLE public.search_table_t2(b int);
CREATE TABLE
gaussdb=#  CREATE TABLE public.search_table_t3(c int);
CREATE TABLE
gaussdb=#  CREATE TABLE public.search_table_t4(d int);
CREATE TABLE
gaussdb=#  CREATE TABLE public.search_table_t5(e int);
CREATE TABLE
```

In the PG_TABLES system catalog, view the tables prefixed with search_table in the public schema.

```
gaussdb=#  SELECT distinct(tablename) FROM pg_tables WHERE SCHEMANAME = 'public' AND
TABLENAME LIKE 'search_table%';
```

The result is as follows:

```
   tablename
-----------------
 search_table_t1
 search_table_t2
```



```
 search_table_t3
 search_table_t4
 search_table_t5
(5 rows)
```

Viewing Database Users

You can use PG_USER to view the list of all users in the database, and view the user ID (**USESYSID**) and permissions.

```
SELECT * FROM pg_user;
 usename | usesysid | usecreatedb | usesuper | usecatupd | userepl |  passwd  | valbegin | valuntil |
respool    | parent | spacelimit | useconfig | no
degroup | tempspacelimit | spillspacelimit | usemonitoradmin | useoperatoradmin | usepolicyadmin
---------+----------+-------------+----------+-----------+---------+----------+----------+----------+--------------
+--------+------------+-----------+---
--------+----------------+-----------------+-----------------+------------------+----------------
 omm     |       10 | t           | t        | t         | t       | ******** |          |          | default_pool |      0 |
|          |
       |              |               | t             | t             | t
```

Viewing and Stopping the Running Query Statements

You can view the running query statements in the **PG_STAT_ACTIVITY** view. You can use the following methods:

Step 1 Set the parameter **track_activities** to **on**.

```
SET track_activities = on;
```

The database obtains the running information about active queries only if the parameter is set to **on**.

Step 2 View the running query statements. View the database names, users performing queries, query status, and the corresponding PID which are connected to the running query statements.

```
SELECT datname, usename, state,pid FROM pg_stat_activity;
 datname  | usename | state  |       pid
----------+---------+--------+-----------------
 testdb | Ruby    | active | 140298793514752
 testdb | Ruby    | active | 140298718004992
 testdb | Ruby    | idle   | 140298650908416
 testdb | Ruby    | idle   | 140298625742592
 testdb | omm | active | 140298575406848
(5 rows)
```

If the **state** column is **idle**, the connection is idle and requires a user to enter a command.

View the query statements that are not in the idle state.

```
SELECT datname, usename, state, pid FROM pg_stat_activity WHERE state != 'idle';
```

Step 3 To cancel queries that have been running for a long time, use the PG_TERMINATE_BACKEND function to end sessions based on the thread ID (corresponding to the PID in **Step 2**).

```
SELECT PG_TERMINATE_BACKEND(140298793514752);
```

If the following information is displayed, the session is successfully terminated:

```
PG_TERMINATE_BACKEND
----------------------
 t
(1 row)
```


If the following information is displayed, a user has terminated the current session:

```
FATAL:  terminating connection due to administrator command
FATAL:  terminating connection due to administrator command
```

NOTE

1. When an initial user uses the PG_TERMINATE_BACKEND function to terminate the backend threads of active sessions, the gsql client is reconnected automatically rather than be logged out. The message "The connection to the server was lost. Attempting reset: Succeeded." is returned. If non-initial users do this operation, the message "The connection to the server was lost. Attempting reset: Failed." is returned. This is because only initial users can log in to the system in password-free mode.

2. If the PG_TERMINATE_BACKEND function is used to end inactive backend threads, and the thread pool is enabled, idle sessions do not have thread IDs and cannot be terminated. In non-thread pool mode, terminated sessions are not automatically reconnected.

```
FATAL:  terminating connection due to administrator command
FATAL:  terminating connection due to administrator command
The connection to the server was lost. Attempting reset: Succeeded.
```

----End

3.1.6 Other Operations

3.1.6.1 Creating and Managing Schemas

Context

Schemas function as models. Schemas allow multiple users to use the same database without interference. In this way, database objects can be organized into logical groups that are easy to manage, and third-party applications can be added to corresponding schemas without causing conflicts. Schema management involves creating a schema, using a schema, deleting a schema, setting a search path for a schema, and setting schema permissions.

Precautions

- GaussDB contains one or more named databases. Users and user groups are shared within the database, but their data is not shared. Any user who has connected to a server can access only the database specified in the connection request.
- A database can have one or more schemas, and a schema can contain tables and other data objects, such as data types, functions, and operators. One object name can be used in different schemas. For example, both schema1 and schema2 can have a table named **mytable**.
- Different from databases, schemas are not isolated. You can access the objects in a schema of the connected database if you have schema permissions. To manage schema permissions, you need to have knowledge about database permissions.
- A schema named with the **PG_** prefix cannot be created because this type of schema is reserved for the database system.
- Each time a new user is created, the system creates a schema with the same name for the new user in the current database. In other databases, such a schema needs to be manually created.

- To reference a table that is not modified with a schema name, the system uses **search_path** to find the schema that the table belongs to. pg_temp and pg_catalog are always the first two schemas to be searched no matter whether or how they are specified in **search_path**. **search_path** is a schema name list, and the first table detected in it is the target table. If no target table is found, an error will be reported. (If a table exists but the schema it belongs to is not listed in **search_path**, the search fails as well.) The first schema in **search_path** is called "current schema." This schema is the first one to be searched. If no schema name is declared, newly created database objects are saved in this schema by default.
- Each database has a pg_catalog schema, which contains system catalogs and all built-in data types, functions, and operators. pg_catalog is a part of the search path and has the second highest search priority. It is searched after the schema of temporary tables and before other schemas specified in **search_path**. This search order ensures that database built-in objects can be found. To use a custom object that has the same name as a built-in object, you can specify the schema of the custom object.

Procedure

- Manage users and their permissions.
  - Create a schema.
    ```
    gaussdb=# CREATE SCHEMA myschema;
    ```
    If the following information is displayed, the schema named **myschema** is successfully created:
    ```
    CREATE SCHEMA
    ```
    To create or access an object in the schema, specify the complete object name, which consists of the schema name and the object names separated by periods (.). for example, **myschema.table**.
  - Create a schema and specify the owner.
    ```
    gaussdb=# CREATE SCHEMA myschema AUTHORIZATION omm;
    ```
    If the following information is displayed, the myschema schema that belongs to the **omm** user is created successfully:
    ```
    CREATE SCHEMA
    ```
- Use a schema.

  If you want to create or access an object in a specified schema, the object name must contain the schema name. To be specific, the name consists of a schema name and an object name, which are separated by a dot (.).
  - Create the **mytable** table in myschema.
    ```
    gaussdb=# CREATE TABLE myschema.mytable(id int, name varchar(20));
    CREATE TABLE
    ```
    To specify the location of an object, the object name must contain the schema name.
  - Query all data of the **mytable** table in myschema.
    ```
    gaussdb=# SELECT * FROM myschema.mytable;
    id | name
    ----+------
    (0 rows)
    ```
- View the search path of a schema.

  You can set **search_path** to specify the sequence of schemas in which objects are searched. The first schema listed in the search path will become the

default schema. If no schema is specified during object creation, the object will be created in the default schema.

- View the search path.

```
gaussdb=# SHOW SEARCH_PATH;
 search_path
----------------
 "$user",public
(1 row)
```

- Set the search path to **myschema, public** (**myschema** will be searched first).

```
gaussdb=# SET SEARCH_PATH TO myschema, public;
SET
```

- Set permissions for a schema.

  By default, a user can only access database objects in their own schema. Only after a user is granted with the usage permission for a schema by the schema owner, the user can access the objects in the schema.

  By granting the CREATE permission for a schema to a user, the user can create objects in this schema. By default, all roles have the USAGE permission in the public schema, but common users do not have the CREATE permission in the public schema. It is insecure for a common user to connect to a specified database and create objects in its public schema. If the common user has the CREATE permission on the public schema, it is advised to:

  - Revoke **PUBLIC**'s permission to create objects in the public schema. In the following command, **public** indicates the schema and **PUBLIC** indicates all roles:

```
gaussdb=# REVOKE CREATE ON SCHEMA public FROM PUBLIC;
REVOKE
```

  - View the current schema.

```
gaussdb=# SELECT current_schema();
 current_schema
----------------
 myschema
(1 row)
```

  - Create user **jack** and grant the usage permission for myschema to the user.

```
gaussdb=# CREATE USER jack IDENTIFIED BY '********';
CREATE ROLE
gaussdb=# GRANT USAGE ON schema myschema TO jack;
GRANT
```

  - Revoke the usage permission for **myschema** from **jack**.

```
gaussdb=# REVOKE USAGE ON schema myschema FROM jack;
REVOKE
```

- Drop a schema.

  - If a schema is empty, that is, it contains no database objects, you can run the **DROP SCHEMA** command. For example, run the following command to drop an empty schema named nullschema:

```
gaussdb=# DROP SCHEMA IF EXISTS nullschema;
DROP SCHEMA
```

  - To drop a schema that is not null, use the keyword CASCADE to drop it and all its objects. For example, run the following command to drop myschema and all its objects:

```
gaussdb=# DROP SCHEMA myschema CASCADE;
DROP SCHEMA
```

  - Delete user **jack**.

```
gaussdb=# DROP USER jack;
DROP ROLE
```


3.1.6.2 Creating and Managing Partitioned Tables

Context

GaussDB supports range partitioned tables, interval partitioned tables, list partitioned tables, and hash partitioned tables.

- Range partitioned table: Data within a certain range is mapped to each partition. The range is determined by the partition key specified when the partitioned table is created. This partitioning mode is most commonly used. The partition key is usually a date. For example, sales data is partitioned by month.
- Interval partitioned table: A special type of range partitioned tables. Compared with range partitioned tables, interval value definition is added. When no matching partition can be found for an inserted record, a partition can be automatically created based on the interval value.
- List partitioned table: Key values contained in the data are stored in different partitions, and the data is mapped to each partition in sequence. The key values contained in the partitions are specified when the partitioned table is created.
- Hash partitioned table: Data is mapped to each partition based on the internal hash algorithm. The number of partitions is specified when the partitioned table is created.

A partitioned table has the following advantages over an ordinary table:

- High query performance: You can specify partitions when querying partitioned tables, improving query efficiency.
- High availability: If a certain partition in a partitioned table is faulty, data in the other partitions is still available.
- Easy maintenance: To fix a partitioned table having a faulty partition, you only need to fix the partition.

To convert an ordinary table to a partitioned table, you need to create a partitioned table and import data to it from the ordinary table. When you design tables, plan whether to use partitioned tables based on service requirements.

Procedure

Example 1: Use the default tablespace.

- Create a partitioned table (assuming that the **tpcds** schema has been created).

```
gaussdb=# CREATE TABLE tpcds.customer_address
(
    ca_address_sk       integer                 NOT NULL   ,
    ca_address_id       character(16)           NOT NULL   ,
    ca_street_number    character(10)                      ,
    ca_street_name      character varying(60)              ,
    ca_street_type      character(15)                      ,
    ca_suite_number     character(10)                      ,
    ca_city             character varying(60)              ,
    ca_county           character varying(30)              ,
    ca_state            character(2)                       ,
    ca_zip              character(10)                      ,
    ca_country          character varying(20)              ,
    ca_gmt_offset       numeric(5,2)                       ,
    ca_location_type    character(20)
```



```
)

PARTITION BY RANGE (ca_address_sk)
(
    PARTITION P1 VALUES LESS THAN(5000),
    PARTITION P2 VALUES LESS THAN(10000),
    PARTITION P3 VALUES LESS THAN(15000),
    PARTITION P4 VALUES LESS THAN(20000),
    PARTITION P5 VALUES LESS THAN(25000),
    PARTITION P6 VALUES LESS THAN(30000),
    PARTITION P7 VALUES LESS THAN(40000),
    PARTITION P8 VALUES LESS THAN(MAXVALUE)
)
ENABLE ROW MOVEMENT;
```

If the following information is displayed, the creation is successful:

```
CREATE TABLE
```

- Insert data.

  Insert data from the **tpcds.customer_address** table to the **tpcds.web_returns_p2** table.

  Suppose the backup table **tpcds.web_returns_p2** of the **tpcds.customer_address** table has been created in the database. You can run the following command to insert the data of the **tpcds.customer_address** table into the backup table **tpcds.web_returns_p2**:

```
gaussdb=# CREATE TABLE tpcds.web_returns_p2
(
    ca_address_sk       integer                 NOT NULL   ,
    ca_address_id       character(16)           NOT NULL   ,
    ca_street_number    character(10)                      ,
    ca_street_name      character varying(60)              ,
    ca_street_type      character(15)                      ,
    ca_suite_number     character(10)                      ,
    ca_city             character varying(60)              ,
    ca_county           character varying(30)              ,
    ca_state            character(2)                       ,
    ca_zip              character(10)                      ,
    ca_country          character varying(20)              ,
    ca_gmt_offset       numeric(5,2)                       ,
    ca_location_type    character(20)
)

PARTITION BY RANGE (ca_address_sk)
(
    PARTITION P1 VALUES LESS THAN(5000),
    PARTITION P2 VALUES LESS THAN(10000),
    PARTITION P3 VALUES LESS THAN(15000),
    PARTITION P4 VALUES LESS THAN(20000),
    PARTITION P5 VALUES LESS THAN(25000),
    PARTITION P6 VALUES LESS THAN(30000),
    PARTITION P7 VALUES LESS THAN(40000),
    PARTITION P8 VALUES LESS THAN(MAXVALUE)
)
ENABLE ROW MOVEMENT;
CREATE TABLE
gaussdb=# INSERT INTO tpcds.web_returns_p2 SELECT * FROM tpcds.customer_address;
INSERT 0 0
```

- Modify the row movement attributes of the partitioned table.

```
gaussdb=# ALTER TABLE tpcds.web_returns_p2 DISABLE ROW MOVEMENT;
ALTER TABLE
```

- Delete a partition.

  Delete partition **P8**.

```
gaussdb=# ALTER TABLE tpcds.web_returns_p2 DROP PARTITION P8;
ALTER TABLE
```


- Add a partition.

  Add partition **P8** and set its range to [40000,MAXVALUE).

  ```
  gaussdb=# ALTER TABLE tpcds.web_returns_p2 ADD PARTITION P8 VALUES LESS THAN
  (MAXVALUE);
  ALTER TABLE
  ```

- Rename a partition.

  – Rename partition **P8** to **P_9**.

  ```
  gaussdb=# ALTER TABLE tpcds.web_returns_p2 RENAME PARTITION P8 TO P_9;
  ALTER TABLE
  ```

  – Rename partition **P_9** to **P8**.

  ```
  gaussdb=# ALTER TABLE tpcds.web_returns_p2 RENAME PARTITION FOR (40000) TO P8;
  ALTER TABLE
  ```

- Query a partition.

  Query partition **P6**.

  ```
  gaussdb=# SELECT * FROM tpcds.web_returns_p2 PARTITION (P6);
  gaussdb=# SELECT * FROM tpcds.web_returns_p2 PARTITION FOR (35888);
  ```

- Delete a partitioned table and its tablespaces.

  ```
  gaussdb=# DROP TABLE tpcds.customer_address;
  DROP TABLE
  gaussdb=# DROP TABLE tpcds.web_returns_p2;
  DROP TABLE
  ```

Example 2: Use a user-defined tablespace (assuming that the **tpcds** schema has been created).

Perform the following operations on a range partitioned table:

- Create a tablespace.

  ```
  gaussdb=# CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace1/tablespace_1';
  gaussdb=# CREATE TABLESPACE example2 RELATIVE LOCATION 'tablespace2/tablespace_2';
  gaussdb=# CREATE TABLESPACE example3 RELATIVE LOCATION 'tablespace3/tablespace_3';
  gaussdb=# CREATE TABLESPACE example4 RELATIVE LOCATION 'tablespace4/tablespace_4';
  ```

  If the following information is displayed, the creation is successful:

  ```
  CREATE TABLESPACE
  ```

- Create a partitioned table.

  ```
  gaussdb=# CREATE TABLE tpcds.customer_address
  (
      ca_address_sk       integer                 NOT NULL   ,
      ca_address_id       character(16)           NOT NULL   ,
      ca_street_number    character(10)                      ,
      ca_street_name      character varying(60)              ,
      ca_street_type      character(15)                      ,
      ca_suite_number     character(10)                      ,
      ca_city             character varying(60)              ,
      ca_county           character varying(30)              ,
      ca_state            character(2)                       ,
      ca_zip              character(10)                      ,
      ca_country          character varying(20)              ,
      ca_gmt_offset       numeric(5,2)                       ,
      ca_location_type    character(20)
  )
  TABLESPACE example1

  PARTITION BY RANGE (ca_address_sk)
  (
      PARTITION P1 VALUES LESS THAN(5000),
      PARTITION P2 VALUES LESS THAN(10000),
      PARTITION P3 VALUES LESS THAN(15000),
      PARTITION P4 VALUES LESS THAN(20000),
      PARTITION P5 VALUES LESS THAN(25000),
      PARTITION P6 VALUES LESS THAN(30000),
      PARTITION P7 VALUES LESS THAN(40000),
  ```



```
    PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;
```

If the following information is displayed, the creation is successful:

```
CREATE TABLE
```

- Insert data.

  Insert data from the **tpcds.customer_address** table to the **tpcds.web_returns_p2** table.

  Suppose the backup table **tpcds.web_returns_p2** of the **tpcds.customer_address** table has been created in the database. You can run the following commands to insert the data of the **tpcds.customer_address** table into the backup table **tpcds.web_returns_p2**:

```
gaussdb=# CREATE TABLE tpcds.web_returns_p2
(
    ca_address_sk       integer                 NOT NULL   ,
    ca_address_id       character(16)           NOT NULL   ,
    ca_street_number    character(10)                      ,
    ca_street_name      character varying(60)              ,
    ca_street_type      character(15)                      ,
    ca_suite_number     character(10)                      ,
    ca_city             character varying(60)              ,
    ca_county           character varying(30)              ,
    ca_state            character(2)                       ,
    ca_zip              character(10)                      ,
    ca_country          character varying(20)              ,
    ca_gmt_offset       numeric(5,2)                       ,
    ca_location_type    character(20)
)
TABLESPACE example1
PARTITION BY RANGE (ca_address_sk)
(
    PARTITION P1 VALUES LESS THAN(5000),
    PARTITION P2 VALUES LESS THAN(10000),
    PARTITION P3 VALUES LESS THAN(15000),
    PARTITION P4 VALUES LESS THAN(20000),
    PARTITION P5 VALUES LESS THAN(25000),
    PARTITION P6 VALUES LESS THAN(30000),
    PARTITION P7 VALUES LESS THAN(40000),
    PARTITION P8 VALUES LESS THAN(MAXVALUE) TABLESPACE example2
)
ENABLE ROW MOVEMENT;
CREATE TABLE
gaussdb=# INSERT INTO tpcds.web_returns_p2 SELECT * FROM tpcds.customer_address;
INSERT 0 0
```

- Modify the row movement attributes of the partitioned table.

```
gaussdb=# ALTER TABLE tpcds.web_returns_p2 DISABLE ROW MOVEMENT;
ALTER TABLE
```

- Delete a partition.

  Delete partition **P8**.

```
gaussdb=# ALTER TABLE tpcds.web_returns_p2 DROP PARTITION P8;
ALTER TABLE
```

- Add a partition.

  Add partition **P8** and set its range to [40000,MAXVALUE).

```
gaussdb=# ALTER TABLE tpcds.web_returns_p2 ADD PARTITION P8 VALUES LESS THAN
(MAXVALUE);
ALTER TABLE
```

- Rename a partition.
  - Rename partition **P8** to **P_9**.

```
gaussdb=# ALTER TABLE tpcds.web_returns_p2 RENAME PARTITION P8 TO P_9;
ALTER TABLE
```


  - Rename partition **P_9** to **P8**.
    ```
    gaussdb=# ALTER TABLE tpcds.web_returns_p2 RENAME PARTITION FOR (40000) TO P8;
    ALTER TABLE
    ```

- Modify the tablespace of a partition.
  - Change the tablespace of partition **P6** to **example3**.
    ```
    gaussdb=#  ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P6 TABLESPACE example3;
    ALTER TABLE
    ```
  - Change the tablespace of partition **P4** to **example4**:
    ```
    gaussdb=#  ALTER TABLE tpcds.web_returns_p2 MOVE PARTITION P4 TABLESPACE example4;
    ALTER TABLE
    ```
- Query a partition.

  Query partition **P6**.
  ```
  gaussdb=# SELECT * FROM tpcds.web_returns_p2 PARTITION (P6);
  gaussdb=# SELECT * FROM tpcds.web_returns_p2 PARTITION FOR (35888);
  ```
- Delete a partitioned table and its tablespaces.
  ```
  gaussdb=# DROP TABLE tpcds.customer_address;
  DROP TABLE
  gaussdb=# DROP TABLE tpcds.web_returns_p2;
  DROP TABLE
  gaussdb=# DROP TABLESPACE example1;
  gaussdb=# DROP TABLESPACE example2;
  gaussdb=# DROP TABLESPACE example3;
  gaussdb=# DROP TABLESPACE example4;
  DROP TABLESPACE
  ```

3.1.6.3 Creating and Managing Indexes

Context

Indexes accelerate data access but increase the processing time of insertion, update, and deletion operations. Therefore, before creating an index, consider whether it is necessary and select the columns where indexes are to be created. You can determine whether to create an index for a table by analyzing the service processing and data use of applications, as well as columns that are frequently used as search criteria or need to be collated.

Indexes are created based on columns in database tables. Therefore, you must correctly identify which columns require indexes. You are advised to create indexes for any of the following columns:

- Columns that are often searched and queried. This speeds up searches.
- Columns that function as primary keys. This enforces the uniqueness of the columns and the data collation structures in organized tables.
- Columns that are often joined. This increases the join efficiency.
- Columns that are often searched by range. The index helps collate data, and therefore the specified ranges are contiguous.
- Columns that often need to be collated. The index helps collate data, reducing the time for a collation query.
- Columns where the WHERE clause is executed frequently. This speeds up condition judgment.
- Columns that often appear after the keywords ORDER BY, GROUP BY, and DISTINCT.

NOTE

- After an index is created, the system automatically determines when to reference it. If the system determines that indexing is faster than sequential scan, the index will be used.
- After an index is successfully created, it must be synchronized with the associated table to ensure new data can be accurately located, which increases the data operation load. Therefore, delete unnecessary indexes periodically.
- Partitioned table indexes are classified into local indexes and global indexes. A local index corresponds to a specific partition, and a global index corresponds to the entire partitioned table.

- When logical replication is enabled, if you need to create a primary key index that contains system columns, you must set the **REPLICA IDENTITY** attribute of the table to **FULL** or use **USING INDEX** to specify a unique, non-local, non-deferrable index that does not contain system columns and contains only columns marked **NOT NULL**.

Procedure

To create a partitioned table, see **Creating and Managing Partitioned Tables**.

- Create an index.
  - Create the local index **tpcds_web_returns_p2_index1** without specifying the partition name.

    ```
    gaussdb=# CREATE INDEX tpcds_web_returns_p2_index1 ON tpcds.web_returns_p2
    (ca_address_id) LOCAL;
    ```

    If the following information is displayed, the creation is successful:

    ```
    CREATE INDEX
    ```

  - Create the local index **tpcds_web_returns_p2_index2** with the partition name specified.

    ```
    gaussdb=# CREATE INDEX tpcds_web_returns_p2_index2 ON tpcds.web_returns_p2
    (ca_address_sk) LOCAL
    (
        PARTITION web_returns_p2_P1_index,
        PARTITION web_returns_p2_P2_index TABLESPACE example3,
        PARTITION web_returns_p2_P3_index TABLESPACE example4,
        PARTITION web_returns_p2_P4_index,
        PARTITION web_returns_p2_P5_index,
        PARTITION web_returns_p2_P6_index,
        PARTITION web_returns_p2_P7_index,
        PARTITION web_returns_p2_P8_index
    ) TABLESPACE example2;
    ```

    If the following information is displayed, the creation is successful:

    ```
    CREATE INDEX
    ```

  - Create the global index **tpcds_web_returns_p2_global_index** for a partitioned table.

    ```
    CREATE INDEX tpcds_web_returns_p2_global_index ON tpcds.web_returns_p2
    (ca_street_number) GLOBAL;
    ```

- Change the tablespace of an index partition.
  - Change the tablespace of index partition **web_returns_p2_P2_index** to **example1**.

    ```
    gaussdb=# ALTER INDEX tpcds.tpcds_web_returns_p2_index2 MOVE PARTITION
    web_returns_p2_P2_index TABLESPACE example1;
    ```

    If the following information is displayed, the modification is successful:

    ```
    ALTER INDEX
    ```


  - Change the tablespace of index partition **web_returns_p2_P3_index** to **example2**.
    ```
    gaussdb=# ALTER INDEX tpcds.tpcds_web_returns_p2_index2 MOVE PARTITION web_returns_p2_P3_index TABLESPACE example2;
    ```
    If the following information is displayed, the modification is successful:
    ```
    ALTER INDEX
    ```

- Rename an index partition.

  Rename the index partition **web_returns_p2_P8_index** to **web_returns_p2_P8_index_new**.
  ```
  gaussdb=# ALTER INDEX tpcds.tpcds_web_returns_p2_index2 RENAME PARTITION web_returns_p2_P8_index TO web_returns_p2_P8_index_new;
  ```
  If the following information is displayed, the rename operation is successful:
  ```
  ALTER INDEX
  ```

- Query indexes.
  - Query all indexes defined by the system and users.
    ```
    gaussdb=# SELECT RELNAME FROM PG_CLASS WHERE RELKIND='i' or RELKIND='I';
    ```
  - Query information about a specified index.
    ```
    gaussdb=# \di+ tpcds.tpcds_web_returns_p2_index2
    ```

- Delete indexes.
  ```
  gaussdb=# DROP INDEX tpcds.tpcds_web_returns_p2_index1;
  gaussdb=# DROP INDEX tpcds.tpcds_web_returns_p2_index2;
  ```
  If the following information is displayed, the deletion is successful:
  ```
  DROP INDEX
  ```

GaussDB supports four methods for creating indexes. For details, see **Table 3-3**.

Table 3-3 Indexing methods

Indexing Method	Description
Unique index	An index that requires the uniqueness of an index attribute or an attribute group. If a table declares unique constraints or primary keys, GaussDB automatically creates unique indexes (or composite indexes) for columns that form the primary keys or unique constraints. Currently, unique indexes can be created only for the B-tree and UB-tree in GaussDB.
Composite index	An index that can be defined for multiple attributes of a table. Currently, the B-tree in GaussDB supports composite indexes.
Partial index	An index that can be created for subsets of a table. This indexing method contains only tuples that meet condition expressions.
Expression index	An index that is built on a function or expression calculated based on one or more attributes of a table. An expression index works only when the queried expression is the same as the created expression.

- Create an ordinary table.


```
gaussdb=# CREATE TABLE tpcds.customer_address_bak AS TABLE tpcds.customer_address;
INSERT 0 0
```

- Create an ordinary index.

  For the **tpcds.customer_address_bak** table, you need to perform the following operations frequently:

  ```
  gaussdb=# SELECT ca_address_sk FROM tpcds.customer_address_bak WHERE ca_address_sk=14888;
  ```

  Generally, the database system needs to scan the **tpcds.customer_address_bak** table row by row to query all matched tuples. If the size of the **tpcds.customer_address_bak** table is large but only a few (possibly zero or one) of the **WHERE** conditions are met, the performance of this sequential scan is poor. If the database system uses an index to maintain the **ca_address_sk** attribute, the database system only needs to search a few tree layers for the matched tuples. This greatly improves data query performance. Furthermore, indexes can improve the update and deletion operation performance in the database.

  Create an index.

  ```
  gaussdb=# CREATE INDEX index_wr_returned_date_sk ON tpcds.customer_address_bak
  (ca_address_sk);
  CREATE INDEX
  ```

- Create a unique index.

  Create a unique index on the **SM_SHIP_MODE_SK** column in the **tpcds.ship_mode_t1** table.

  ```
  gaussdb=# CREATE UNIQUE INDEX ds_ship_mode_t1_index1 ON
  tpcds.ship_mode_t1(SM_SHIP_MODE_SK);
  ```

- Create a composite index.

  To frequently query records with **ca_address_sk** being **5050** and **ca_street_number** smaller than **1000** in the **tpcds.customer_address_bak** table, run the following command:

  ```
  gaussdb=# SELECT ca_address_sk,ca_address_id FROM tpcds.customer_address_bak WHERE
  ca_address_sk = 5050 AND ca_street_number < 1000;
  ```

  Define a composite index on the **ca_address_sk** and **ca_street_number** columns.

  ```
  gaussdb=# CREATE INDEX more_column_index ON
  tpcds.customer_address_bak(ca_address_sk ,ca_street_number );
  CREATE INDEX
  ```

- Create a partial index.

  If you only want to find records with **ca_address_sk** being **5050**, you can create a partial index to facilitate your query.

  ```
  gaussdb=# CREATE INDEX part_index ON tpcds.customer_address_bak(ca_address_sk) WHERE
  ca_address_sk = 5050;
  CREATE INDEX
  ```

- Create an expression index.

  Frequently query records with **ca_street_number** smaller than **1000**.

  ```
  gaussdb=# SELECT * FROM tpcds.customer_address_bak WHERE trunc(ca_street_number) < 1000;
  ```

  The following expression index can be created for this query task:

  ```
  gaussdb=# CREATE INDEX para_index ON tpcds.customer_address_bak (trunc(ca_street_number));
  CREATE INDEX
  ```

- Delete the **tpcds.customer_address_bak** table.

  ```
  gaussdb=# DROP TABLE tpcds.customer_address_bak;
  DROP TABLE
  ```


3.1.6.4 Creating and Managing Views

Context

If some columns in one or more tables in a database are frequently searched for, an administrator can define a view for these columns, and then users can directly access these columns in the view without entering search criteria.

Different from a base table, a view is a virtual table. Only view definition is stored in the database and view data is not. The data is stored in a base table. If data in the base table changes, the data in the view changes accordingly. A view is like a window through which users can know their interested data and data changes in the database. A view is triggered every time it is referenced.

Managing Views

- Create a view.

  Create the **MyView** view. In the command, **tpcds.web_returns** indicates the created user table that contains the **wr_refunded_cash** integer column.

  ```
  gaussdb=# CREATE OR REPLACE VIEW MyView AS SELECT * FROM tpcds.web_returns WHERE trunc(wr_refunded_cash) > 10000;
  CREATE VIEW
  ```

  NOTE

  The **OR REPLACE** parameter in this command is optional. It indicates that if the view exists, the new view will replace the existing view.

- Query a view.

  Query **MyView**.

  ```
  gaussdb=# SELECT * FROM MyView;
  ```

- Query views of the current user.

  ```
  gaussdb=# SELECT * FROM my_views;
  ```

- Query all views.

  ```
  gaussdb=# SELECT * FROM adm_views;
  ```

- Query details about a specified view.

  View details about **MyView**.

  ```
  gaussdb=# \d+ MyView
                      View "PG_CATALOG.MyView"
   Column  |         Type          | Modifiers | Storage  | Description
  ----------+-----------------------+-----------+----------+-------------
   USERNAME | CHARACTER VARYING(64) |           | extended |
  View definition:
   SELECT PG_AUTHID.ROLNAME::CHARACTER VARYING(64) AS USERNAME
     FROM PG_AUTHID;
  ```

- Delete a view.

  Delete **MyView**.

  ```
  gaussdb=# DROP VIEW MyView;
  DROP VIEW
  ```

3.1.6.5 Creating and Managing Sequences

Context

A sequence is a database object that generates unique integers. Sequence numbers are generated according to a certain rule. Sequences are unique because they increase automatically. This is why they are often used as primary keys.

You can create a sequence for a column in either of the following methods:

- Set the data type of a column to **sequence integer**. A sequence will be automatically created by the database for this column.
- Run the **CREATE SEQUENCE** statement to create a sequence. Set the initial value of the nextval('sequence_name') function to the default value of a column.

Procedure

Method 1: Set the data type of a column to a sequence integer. For example:

```
gaussdb=# CREATE TABLE T1
(
   id    serial,
   name  text
);
```

If the following information is displayed, the creation is successful:

```
CREATE TABLE
```

Method 2: Create a sequence and set the initial value of the nextval('sequence_name') function to the default value of a column.

1. Create a sequence.

   ```
   gaussdb=# CREATE SEQUENCE seq1 cache 100;
   ```

   If the following information is displayed, the creation is successful:

   ```
   CREATE SEQUENCE
   ```

2. Set the default value of a column so that the column has a unique identification attribute.

   ```
   gaussdb=# CREATE TABLE T2
   (
      id   int not null default nextval('seq1'),
      name text
   );
   ```

   If the following information is displayed, the default value has been specified:

   ```
   CREATE TABLE
   ```

3. Associate a sequence with a column.

   Associate a sequence with a specified column included in a table. In this way, the sequence will be deleted when you delete its associated column or the table where the column belongs to.

   ```
   gaussdb=# ALTER SEQUENCE seq1 OWNED BY T2.id;
   ```

   If the following information is displayed, the operation is successful:

   ```
   ALTER SEQUENCE
   ```

NOTE

The preceding methods are similar, except that the second method specifies cache for the sequence. A sequence having cache defined has inconsecutive values (such as 1, 4, and 5) and cannot maintain the order of its values. After the dependent column of a sequence has been specified, once the sequence is deleted, the sequence of the dependent will be deleted. A sequence shared by multiple columns is not forbidden in a database, but you are advised not to do that.

In the current version, you can specify the auto-increment column or set the default value of a column to **nextval('seqname')** when defining a table. You cannot add an auto-increment column or a column whose default value is **nextval('seqname')** to an existing table.

3.1.6.6 Creating and Managing Scheduled Jobs

Context

Time-consuming jobs, such as summarizing statistics or synchronizing data from another database, affect service performance if they are performed during the daytime and incur overtime hours if performed at night. To solve this problem, the GaussDB Kernel database is compatible with the scheduled job function in the A database. You can create scheduled jobs that are automatically triggered to reduce O&M workload.

This function calls interfaces provided by the **DBE_TASK** package to create scheduled jobs, execute jobs automatically, delete jobs, modify job attributes (including job ID, the enabled/disabled status of a job, job triggering time, triggering interval, and job contents), and implement other functions.

Managing Scheduled Jobs

Step 1 Create a test table.

```
gaussdb=# CREATE TABLE test(id int, time date);
```

If the following information is displayed, the creation is successful:

```
CREATE TABLE
```

Step 2 Create a customized stored procedure.

```
gaussdb=# CREATE OR REPLACE PROCEDURE PRC_JOB_1()
AS
N_NUM integer :=1;
BEGIN
FOR I IN 1..1000 LOOP
INSERT INTO test VALUES(I,SYSDATE);
END LOOP;
END;
/
```

If the following information is displayed, the creation is successful:

```
CREATE PROCEDURE
```

Step 3 Create a task.

- Create a job with unspecified **job_id** and execute the **PRC_JOB_1** stored procedure every minute.

  ```
  gaussdb=# call dbe_task.submit('call public.prc_job_1(); ', sysdate, 'interval ''1 minute'''', :a);
   id
  -----
  1
  (1 row)
  ```

- Specify **job_id** to create a job. The value of **job_id** ranges from 1 to 32767.

  ```
  gaussdb=# call dbe_task.id_submit(1,'call public.prc_job_1(); ', sysdate, 'interval ''1 minute''');
  id_submit
  ---------


  (1 row)
  ```

Step 4 View details of jobs created by the current user.

```
gaussdb=# select job,dbname,start_date,last_date,this_date,next_date,broken,status,interval,failures,what
from my_jobs;
job | dbname |     start_date      |        last_date         |        this_date         |      next_date      | broken |
status |      interval       | failures |          what
-----+--------+---------------------+--------------------------+--------------------------+---------------------
```



```
+--------+--------+---------------------+----------+--------------------------
1 | testdb   | 2017-07-18 11:38:03 | 2017-07-18 13:53:03.607838 | 2017-07-18 13:53:03.607838 | 2017-07-18
13:54:03 | n      | s      | interval '1 minute' |        0 | call public.prc_job_1();
(1 row)
```

Step 5 Stop a job.

```
gaussdb=# call dbe_task.finish(1,true);
finish
--------


(1 row)
```

Step 6 Start a job.

```
gaussdb=# call dbe_task.finish(1,false);
finish
--------


(1 row)
```

Step 7 Modify job attributes.

- Modify the **Next_date** parameter of a job.

  ```
  -- Set Next_date to execute Job1 1 hour later.
  gaussdb=# call dbe_task.next_date(1, sysdate+1.0/24);
  next_date
  -----------


  (1 row)
  ```

- Modify the **Interval** parameter of a job.

  ```
  -- Set Interval of Job1 to 1.
  gaussdb=# call dbe_task.interval(1,'sysdate + 1.0/24');
   interval
  ----------


  (1 row)
  ```

- Modify the **What** parameter of a job.

  ```
  -- Set What to insert into public.test values(333, sysdate+5); for Job1.
  gaussdb=# call dbe_task.content(1,'insert into public.test values(333, sysdate+5);');
   content
  ---------


  (1 row)
  ```

- Modify **Next_date**, **Interval**, and **What** parameters of a job.

  ```
  gaussdb=# call dbe_task.update(1, 'call public.prc_job_1();', sysdate, 'interval ''1 minute''');
   update
  --------


  (1 row)
  ```

Step 8 Delete a job.

```
gaussdb=# call dbe_task.cancel(1);
 cancel
---------


(1 row)
```

Step 9 View the job execution status.

If a job fails to be automatically executed (that is, the value of **job_status** is **'f'**), contact the administrator to view the gs_log run log to view the failure information of the job.

From **detail error msg**, you can see the failure causes.

```
LOG:  Execute Job Detail:
        job_id: 1
```



```
what: call public.test();
start_date: 2017-07-19 23:30:47.401818
job_status: failed
detail error msg: relation "test" does not exist
end_date: 2017-07-19 23:30:47.401818
next_run_date: 2017-07-19 23:30:56.855827
```

Step 10 Set job permissions.

- During the creation of a job, the job is bound to the user and database that created the job. Accordingly, the user and database are added to **dbname** and **log_user** columns in the **pg_job** system catalog, respectively.
- If the current user is a database administrator, system administrator, or the user (**log_user** of **pg_job**) who created the job, the user has permissions to delete or modify job parameters using the **remove**, **change**, **next_data**, **what**, or **interval** parameter. Otherwise, the system displays a message indicating that the user has no permissions to perform operations on this job.
- If the current database is the one that created a job, (that is, **dbname** in **pg_job**), you can delete or modify parameter settings of the job using the **cancel**, **update**, **next_data**, **content**, or **interval** parameter.
- When deleting the database that created a job, (that is, **dbname** in **pg_job**), the system automatically deletes the job records of the database.
- When deleting the user who created a job, (that is, **log_user** in **pg_job**), the system automatically deletes the job records of the user.

Step 11 Manage job concurrency.

You can configure the GUC parameter **job_queue_processes** to adjust the number of jobs running at the same time.

- Setting **job_queue_processes** to **0** indicates that the scheduled job function is disabled and all jobs will not be executed.
- Setting **job_queue_processes** to a value that is greater than **0** indicates that the scheduled job function is enabled and this value is the maximum number of jobs that can be concurrently processed.

Too many concurrent jobs consume many system resources, so you need to set the number of concurrent jobs to be processed. If the current number of concurrent jobs reaches the value of **job_queue_processes** and some of them expire, these jobs will be postponed to the next polling period. Therefore, you are advised to set the polling interval (the **Interval** parameter of the **submit** interface) based on the execution duration of each job to avoid the problem that jobs in the next polling period cannot be properly processed because of overlong job execution time.

Note: For database instances that do not use jobs, set **job_queue_processes** to **0** to disable job functions to reduce the resource consumption.

----End

4 Development and Design Proposal

4.1 Overview

This section describes the design specifications for database modeling and application development. Modeling based on these specifications can better fit the processing architecture of GaussDB and output more efficient service SQL code.

The meaning of "Proposal" and "Notice" in this section is as follows:

- **Proposal**: Design rules. Services compliant with the rules can run efficiently, and those violating the rules may have low performance or logic errors.
- **Notice**: Details requiring attention during service development. This term identifies SQL behavior that complies with SQL standards but users may have misconceptions about, and default behavior that users may be unaware of in a program.

4.2 Database Object Naming Conventions

Database object names must meet the following requirements:

- The length of a table of an identifier cannot exceed 63 bytes.
- An identifier starts with a letter or underscore (_) and can contain letters, digits, underscores (_), dollar signs ($), and number signs (#).
- If an identifier is enclosed in backquotes (``) in B-compatible mode or double quotation marks (""), any combination of valid characters can be used, for example, "123gs_column".
- Identifiers are case-insensitive. They are case-sensitive only when they are enclosed in backquotes (``) in B-compatible mode or double quotation marks ("").
- The gsql shortcut commands (except **\sf**) cannot be used to query the object names enclosed in backquotes.
- Do not use reserved or non-reserved keywords to name database objects.

> **NOTE**
>
> You can use the select * from pg_get_keywords() query GaussDB keyword or view the keyword in **Keywords**.

- Do not use a string enclosed in double quotation marks ("") to define the database object name, unless you need to specify its capitalization. Case sensitivity of database object names makes problem location difficult.
- Use the same naming format for database objects.
  - In a system undergoing incremental development or service migration, you are advised to comply with its historical naming conventions.
  - You are advised to use multiple words separated with underscores (_).
  - You are advised to use intelligible names and common acronyms or abbreviations for database objects. Acronyms or abbreviations that are generally understood are recommended. For example, you can use English words or Chinese pinyin indicating actual business terms. The naming format should be consistent within a database instance.
  - A variable name must be descriptive and meaningful. It must have a prefix indicating its type.
- The name of a table object should indicate its main characteristics, for example, whether it is an ordinary, temporary, or unlogged table.
  - An ordinary table name should indicate the business relevant to a dataset.
  - Temporary tables are named in the format of **tmp**_*Suffix*.
  - Unlogged tables are named in the format of **ul**_*Suffix*.
  - Foreign tables are named in the format of **f**_*Suffix*.
  - Do not create database objects whose names start with **redis**_.
  - Do not create database objects whose names start with **mlog**_ or **matviewmap**_.
  - Do not create database objects whose names start with **gs_role**_.
- The name of a table object shall not exceed 63 bytes. If the length of a table name exceeds this value, the kernel truncates the table name. As a result, the table name is inconsistent with the configured value. In addition, characters may be truncated in different character sets and unexpected characters may appear.

4.3 Database Object Design

4.3.1 Database and Schema Design

In GaussDB, services can be isolated by databases and schemas. Databases share little resources and cannot directly access each other. Connections to and permissions on them are also isolated. Schemas share more resources than databases do. User permissions on schemas and subordinate objects can be controlled using the GRANT and REVOKE syntax.

- You are advised to use schemas to isolate services for convenience and resource sharing.

- It is recommended that system administrators create schemas and databases and then assign required permissions to users.

Database Design

- Create databases as required by your service. Do not use the default **postgres** database of a database instance.
- In a database instance, it is recommended that the number of user-defined databases be 3 and no more than 10. If there are too many user-defined databases, O&M operations, such as upgrade and backup, will be inefficient.
- To make your database compatible with most characters, you are advised to use the UTF-8 encoding when creating a database.
- When you create a database, exercise caution when you set **ENCODING** and **DBCOMPATIBILITY** configuration items. GaussDB supports the A-, B-, C-, PG-, and M-compatible modes, which are compatible with the Oracle syntax, MySQL syntax, Teradata syntax, Postgres syntax, and M-compatible syntax, respectively. The syntax behavior varies according to the compatibility mode. By default, the A-compatible mode is used.
- By default, a database owner has all permissions for all objects in the database, including the deletion permission. Exercise caution when deleting a permission.

Schema Design

- It is recommended that the number of schemas in the actual user environment be no more than 100. If there are too many schemas in a database, operations that depend on the number of schemas, such as gs_dump, becomes slow.
- To let a user access an object in a schema, assign the usage permission and the permissions for the object to the user, unless the user has the sysadmin permission or is the schema owner.
- To let a user create an object in the schema, grant the CREATE permission for the schema to the user.
- By default, a schema owner has all permissions for all objects in the schema, including the deletion permission. Exercise caution when deleting a permission.

4.3.2 Table Design

Generally, well-designed table must comply with the following rules:

- Reduce the amount of data to be scanned. You can use the pruning mechanism of a partitioned table.
- Minimize random I/Os. Through clustering, you can sequentially store hot data, converting random I/O to sequential I/O to reduce the cost of I/O scanning.

Selecting a Partitioning Mode

Comply with the following rules to partition a table containing a large amount of data:

- Create partitions on columns that indicate certain ranges, such as dates and regions.
- A partition name should show the data characteristics of a partition. For example, its format can be *Keyword*+*Range* characteristics.
- Set the upper limit of a partition to **MAXVALUE** to prevent data overflow.

Table 4-1 Table partitioning modes and scenarios

Partitioning Mode	Description
Range	Table data is partitioned by range.
Interval	Table data is partitioned by range. If the data exceeds the range, a new partition is automatically created based on the interval.
List	Table data is partitioned by a specified column based on a specific value.
Hash	Table data is partitioned by hash.

The example of defining a partitioned table is as follows:

```
-- Create a range partitioned table.
CREATE TABLE staffS_p1
(
  staff_ID       NUMBER(6) not null,
  FIRST_NAME     VARCHAR2(20),
  LAST_NAME      VARCHAR2(25),
  EMAIL          VARCHAR2(25),
  PHONE_NUMBER   VARCHAR2(20),
  HIRE_DATE      DATE,
  employment_ID  VARCHAR2(10),
  SALARY         NUMBER(8,2),
  COMMISSION_PCT NUMBER(4,2),
  MANAGER_ID     NUMBER(6),
  section_ID     NUMBER(4)
)
PARTITION BY RANGE (HIRE_DATE)
(
   PARTITION HIRE_19950501 VALUES LESS THAN ('1995-05-01 00:00:00'),
   PARTITION HIRE_19950502 VALUES LESS THAN ('1995-05-02 00:00:00'),
   PARTITION HIRE_maxvalue VALUES LESS THAN (MAXVALUE)
);

-- Create an interval partitioned table. The table has two initial partitions. When data that is not in the partition range is inserted, another partition is automatically added.
CREATE TABLE sales
(prod_id NUMBER(6),
 cust_id NUMBER,
 time_id DATE,
 channel_id CHAR(1),
 promo_id NUMBER(6),
 quantity_sold NUMBER(3),
 amount_sold NUMBER(10,2)
)
PARTITION BY RANGE (time_id)
INTERVAL('1 day')
( PARTITION p1 VALUES LESS THAN ('2019-02-01 00:00:00'),
  PARTITION p2 VALUES LESS THAN ('2019-02-02 00:00:00')
);
```



```
-- Create a list partitioned table.
CREATE TABLE test_list (col1 int, col2 int)
partition by list(col1)
(
partition p1 values (2000),
partition p2 values (3000),
partition p3 values (4000),
partition p4 values (5000)
);

-- Create a hash partitioned table.
CREATE TABLE test_hash (col1 int, col2 int)
partition by hash(col1)
(
partition p1,
partition p2
);
```

For details about the table partition syntax, see **CREATE TABLE PARTITION**.

4.3.3 Column Design

Selecting a Data Type

To improve query efficiency, comply with the following rules when designing columns:

- Use the most efficient data types allowed.

  If all of the following number types provide the required service precision, they are recommended in descending order of priority: integer, floating point, and numeric.
- In tables that are logically related, columns having the same meaning should use the same data type.
- For string data, you are advised to use variable-length strings and specify the maximum length. Ensure that the specified maximum length is greater than the maximum number of characters to be stored. Otherwise, an error is reported, causing service interruption. You are advised not to use CHAR(n), BPCHAR(n), NCHAR(n), or CHARACTER(n), unless you know that the string length is fixed.

  For details about string types, see **Common String Types**.

Common String Types

Every column requires a data type suitable for its data characteristics. **Character Types** lists common string types in GaussDB.

4.3.4 Constraint Design

DEFAULT and NULL Constraints

- If all the column values can be obtained from services, you are advised not to use the DEFAULT constraint. Otherwise, unexpected results will be generated during data loading.
- Add the NOT NULL constraint for columns that do not have NULL values. The optimizer automatically optimizes the columns in certain scenarios.
- Explicitly name all constraints excluding NOT NULL and DEFAULT.

Unique Constraints

- The constraint name should indicate that it is a unique constraint, for example, **UNI***column name*.

Primary Key Constraints

- The constraint name should indicate that it is a primary key constraint, for example, **PK***Included columns*.

Check Constraints

- The constraint name should indicate that it is a check constraint, for example, **CK***Included columns*.

4.3.5 View and Joined Table Design

View Design

- Do not nest views unless they have strong dependency on each other.
- Try to avoid collation operations in a view definition.

Joined Table Design

- Minimize joined columns across tables.
- Use the same data type for joined columns.
- The names of joined columns should indicate their relationship. For example, they can use the same name.

4.4 Tool Interconnection

4.4.1 JDBC Configuration

Currently, third-party tools are connected to GaussDB through JDBC. This section describes the precautions for configuring the tool.

Connection Parameters

- When a third-party tool connects to GaussDB through JDBC, JDBC sends a connection request to GaussDB. By default, the following configuration parameters are added. For details, see the implementation of the ConnectionFactoryImpl class in the JDBC code.

```
params = {
          { "user", user },
          { "database", database },
          { "client_encoding", "UTF8" },
          { "DateStyle", "ISO" },
          { "extra_float_digits", "3" },
          { "TimeZone",  createPostgresTimeZone() },
        };
```

  These parameters may cause the JDBC and gsql clients to display inconsistent data, for example, date data display mode, floating point precision representation, and timezone.

If the result is not as expected, you are advised to explicitly set these parameters in the Java connection setting.

When the database is connected through JDBC, **extra_float_digits** is set to **3**. When the database is connected using gsql, **extra_float_digits** is set to **0**. As a result, the precision of the same data displayed in JDBC clients may be different from that displayed in gsql clients.

- In precision-sensitive scenarios, the numeric type is recommended.
- When JDBC connects to the database, ensure that the following three time zones are the same:
  - Time zone of the host where the JDBC client is located
  - Time zone of the host where the GaussDB database instance is located.
  - Time zone used during GaussDB database instance configuration.

  NOTE

  For details about how to set the time zone, contact the administrator.

fetchsize

To use **fetchsize** in applications, disable **autocommit**. Enabling the **autocommit** switch makes the **fetchsize** configuration invalid.

autocommit

You are advised to enable **autocommit** in the code for connecting to GaussDB by the JDBC. If **autocommit** needs to be disabled to improve performance or for other purposes, applications need to ensure that transactions are committed. For example, explicitly commit translations after specifying service SQL statements. Particularly, ensure that all transactions are committed before the client exits.

Connection Releasing

- You are advised to use connection pools to limit the number of connections from applications. You are advised not to connect to a database each time an SQL statement is executed.
- After an application completes its jobs, disconnect it from GaussDB to release occupied resources. You are advised to set the session timeout interval in the jobs.
- Reset the session environment before releasing connections to the JDBC connection pool. Otherwise, historical session information may cause object conflicts.
  - If GUC parameters are set in the connection, run **SET SESSION AUTHORIZATION DEFAULT;RESET ALL;** to clear the connection status before you return the connection to the connection pool.
  - If a temporary table is used, delete the temporary table before you return the connection to the connection pool.

CopyManager

In the scenario where the ETL tool is not used and real-time data import is required, it is recommended that you use the CopyManager interface driven by the GaussDB JDBC to import data in batches during application development.

4.5 SQL Compilation

DDL

- In GaussDB, you are advised to perform DDL operations (such as table creation and COMMENT) in a unified manner. Prevent DDL operations in batch processing jobs so that performance is not affected.
- Perform the TRUNCATE operation immediately after the unlogged table is used because GaussDB cannot ensure the security of unlogged tables in abnormal scenarios.
- Suggestions on the storage mode of temporary and unlogged tables are the same as those on base tables.
- The total length of an index column cannot exceed 50 bytes. Otherwise, the index size will increase greatly, resulting in large storage cost and low index performance.
- Do not delete objects using DROP...CASCADE, unless the dependency between objects is specified. Otherwise, the objects may be deleted by mistake.

Data Loading and Unloading

- Explicitly set the inserted column list in the INSERT statement. For example:
  ```
  INSERT INTO task(name,id,comment) VALUES ('task1','100','100th task');
  ```
- After data is imported to the database in batches or the data increment reaches the threshold, you are advised to analyze tables to prevent the execution plan from being degraded due to inaccurate statistics.
- To clear all data in a table, you are advised to use TRUNCATE TABLE instead of DELETE TABLE. DELETE TABLE is not efficient and cannot release disk space occupied by the deleted data.

Type Conversion

- Convert data types explicitly. If you perform implicit conversion, the result may differ from expected.
- During data query, explicitly specify the data type for constants, and do not attempt to perform any implicit data type conversion.
- If **sql_compatibility** is set to **A**, null strings will be automatically converted to **NULL** during data import. If null strings need to be reserved, set **sql_compatibility** to **C**.

Query Operation

- Do not return a large number of result sets to a client except the ETL program. If a large result set is returned, consider modifying your service design.

- Perform DDL and DML operations encapsulated in transactions. For example, operations such as TRUNCATE TABLE, UPDATE TABLE, DELETE TABLE, and DROP TABLE cannot be restored once they are committed. You are advised to encapsulate such operations in transactions so that you can roll back the operations if necessary.
- During query compilation, you are advised to list all columns to be queried and avoid using SELECT *. Doing so reduces output lines, improves query performance, and avoids the impact of adding or deleting columns on front-end service compatibility.
- During table object access, add the schema prefix to the table object to avoid accessing an unexpected table due to schema switchover.
- The cost of joining more than three tables or views, especially full joins, is difficult to be estimated. You are advised to use the WITH TABLE AS statement to create interim tables to improve the readability of SQL statements.
- Avoid using Cartesian products or full joins. Cartesian products and full joins will result in a sharp expansion of result sets and poor performance.
- Only IS NULL and IS NOT NULL can be used to determine NULL value comparison results. If any other method is used, NULL is returned. For example, **NULL** instead of expected Boolean values is returned for **NULL<>NULL**, **NULL=NULL**, and **NULL<>1**.
- Do not use count(col) instead of count(*) to count the total number of records in a table. **count(*)** counts the NULL value (actual rows) while **count(col)** does not.
- While executing count(col), the number of **NULL** record rows is counted as 0. While executing sum(col), **NULL** is returned if all records are **NULL**. If not all the records are **NULL**, the number of **NULL** record rows is counted as 0.
- To count multiple columns using count(), column names must be enclosed in parentheses, for example, count((col1, col2, col3)). When multiple columns are used to count the number of NULL record rows, a row is counted even if all the selected columns are NULL. The result is the same as that when count(*) is executed.
- NULL records are not counted when count(distinct col) is used to calculate the number of non-NULL columns that are not repeated.
- If all statistical columns are NULL when count(distinct (col1,col2,...)) is used to count the number of unique values in multiple columns, Null records are also counted, and the records are considered the same.
- Use the connection operator || to replace the concat function for string connection, because the concat function needs to query type tables and function tables, which slows down the basic performance. In addition, because the concat output is related to the data type, the execution plan generated by the concat function cannot calculate results in advance. As a result, the query performance severely deteriorates.
- Use the time-related macros listed in **Table 1** to replace the now function and obtain the current time, because the execution plan generated by the now function cannot be pushed down to disks. As a result, the query performance severely deteriorates.

Table 4-2 Time-related macros

Macro Name	Description	Example
CURRENT_DATE	Obtains the current date, excluding the hour, minute, and second details.	gaussdb=# SELECT CURRENT_DATE; date ------------ 2018-02-02 (1 row)
CURRENT_TIME	Obtains the current time, excluding the year, month, and day.	gaussdb=# SELECT CURRENT_TIME; timetz -------------------- 00:39:34.633938+08 (1 row)
CURRENT_TIMESTAMP(n)	Obtains the current date and time, including the year, month, day, hour, minute, second, and time zone. **NOTE** *n* indicates the number of digits after the decimal point in the time string.	gaussdb=# SELECT CURRENT_TIMESTAMP(6); timestamptz ------------------------------- 2018-02-02 00:39:55.231689+08 (1 row)

- Do not use scalar subquery statements. A scalar subquery is a subquery in the output list of the SELECT statement. In the following example, "SELECT COUNT(*) FROM films f WHERE f.did = s.id" is a scalar subquery statement:
  ```
  SELECT id, (SELECT COUNT(*) FROM films f WHERE f.did = s.id) FROM staffs_p1 s;
  ```
  Scalar subqueries often result in query performance deterioration. During application development, scalar subqueries need to be converted into equivalent table associations based on the service logic.
- In WHERE clauses, the filter conditions should be collated. The condition that few records are selected for reading (the number of filtered records is small) is listed at the beginning.
- Filter conditions in WHERE clauses should comply with unilateral rules, that is, to place the column name on one side of a comparison operator. In this way, the optimizer automatically performs pruning optimization in some scenarios. The format is *col op expression*, where *col* indicates a table column, *op* indicates a comparison operator, such as = and >, and *expression* indicates an expression that does not contain a column name. For example:
  ```
  SELECT id, from_image_id, from_person_id, from_video_id FROM face_data WHERE current_timestamp(6) - time < '1 days'::interval;
  ```
  The modification is as follows:
  ```
  SELECT id, from_image_id, from_person_id, from_video_id FROM face_data where time > current_timestamp(6) - '1 days'::interval;
  ```
- Do not perform unnecessary collation operations. Collation requires a large amount of memory and CPU. If service logic permits, ORDER BY and LIMIT can be combined to reduce resource overhead. By default, GaussDB performs collation by ASC & NULL LAST.

- When the ORDER BY clause is used for collation, specify collation modes (**ASC** or **DESC**), and use NULL FIRST or NULL LAST for NULL record sorting.
- Do not rely on only the LIMIT clause to return the result set displayed in a specific sequence. Combine ORDER BY and LIMIT clauses if some specific result sets are returned, and use OFFSET to skip specific results if necessary.
- If the service logic is accurate, you are advised to use UNION ALL instead of UNION.
- If a filter condition contains only an OR expression, convert the OR expression to UNION ALL to improve performance. SQL statements that use OR expressions cannot be optimized, resulting in slow execution. For example, the conversion of the following statements:

  ```
  SELECT * FROM scdc.pub_menu
  WHERE (cdp= 300 AND inline=301) OR (cdp= 301 AND inline=302) OR (cdp= 302 ANDinline=301);
  ```

  Convert the statement to the following:

  ```
  SELECT * FROM scdc.pub_menu
  WHERE (cdp= 300 AND inline=301)
  union all
  SELECT * FROM scdc.pub_menu
  WHERE (cdp= 301 AND inline=302)
  union all
  SELECT * FROM tablename
  WHERE (cdp= 302 AND inline=301)
  ```

- If an IN(val1, val2, val3…) expression contains a large number of columns, you are advised to replace it with the IN (VALUES (va1), (val2),(val3)…) statement. The optimizer will automatically convert the IN constraint into a non-correlated subquery to improve the query performance.
- Use (NOT) EXIST instead of (NOT) IN when associated columns do not contain null values. For example, in a query statement, if the **T1.C1** column does not contain any **NULL** value, add the **NOT NULL** constraint to the **T1.C1** column, and then rewrite the statements.

  ```
  SELECT * FROM T1 WHERE T1.C1 NOT IN (SELECT T2.C2 FROM T2);
  ```

  Rewrite the statement as follows:

  ```
  SELECT * FROM T1 WHERE NOT EXISTS (SELECT  * FROM T2 WHERE T1.C1=T2.C2);
  ```

  NOTE

  - If the value of the **T1.C1** column is not **NOT NULL**, the preceding rewriting cannot be performed.
  - If the **T1.C1** column is the output of a subquery, check whether the output is **NOT NULL** based on the service logic.
- Use cursors instead of the LIMIT OFFSET syntax to perform pagination queries to avoid resource overheads caused by multiple executions. A cursor must be used in a transaction, and you must disable the cursor and commit the transaction once the query is finished.

5 Application Development Guide

5.1 Development Specifications

If the connection pool mechanism is used during application development, comply with the following specifications:

- If GUC parameters are set in the connection, run **SET SESSION AUTHORIZATION DEFAULT;RESET ALL;** to clear the connection status before you return the connection to the connection pool.
- If a temporary table is used, delete the temporary table before you return the connection to the connection pool.

If you do not do so, the connection state in the connection pool will remain, which affects subsequent operations using the connection pool.

Table 5-1 describes the compatibility of application development drivers.

Table 5-1 Compatibility Description

Driver	Compatibility Description
JDBC, Go, ODBC, libpq, Psycopg, and ecpg	The new drivers are forward compatible with the database. To use the new features added to the driver and database, you must upgrade the database.

NOTICE

- Setting **behavior_compat_options** to **'proc_outparam_override'** is applicable only in A-compatible mode.
- In principle, you need to set the compatibility parameter after the database creation, instead of switching the parameters when using the database.
- The JDBC driver must be upgraded to that maps to GaussDB Kernel 503.1.0 or later if the following features are used:
  1. The fully-encrypted memory decryption emergency channel is required.
  2. JDBC is required to use user-defined nested types such as record, array, and tableof.
  3. The **s2** compatibility parameter is enabled and the validity check of **sessiontimezone** is set.

If the driver is used in a multi-thread environment:

The JDBC driver is not thread-safe and does not guarantee that the connection methods are synchronized. The caller synchronizes the calls to the driver.

5.2 Obtaining the Driver Package

Obtaining the Driver Package

Download particular packages listed in **Table 5-2** based on the version of your instance.

Table 5-2 Driver package download list

Version	Download Address
8.x	**Driver package** **Verification package for the driver package**

To prevent a software package from being tampered with during transmission or storage, download the corresponding verification package and perform the following steps to verify the software package:

1. Upload the software package and verification package to the same directory on a Linux VM.
2. Run the following command to verify the integrity of the software package:

   cat *GaussDB_driver.zip.sha256* **| sha256sum --check**

   If **OK** is displayed in the command output, the verification is successful.

   ```
   GaussDB_driver.zip: OK
   ```


5.3 Development Based on JDBC

Java Database Connectivity (JDBC) is a Java API for running SQL statements. It provides unified access APIs for different relational databases, based on which applications process data. The GaussDB library supports JDBC 4.2 and requires JDK 1.8 for code compiling. It does not support JDBC-ODBC bridge.

5.3.1 JDBC Packages, Driver Classes, and Environment Classes

JDBC Package

Obtain the package from the release package **GaussDB-Kernel**_*Database version number*_*OS version number*_**64bit_Jdbc.tar.gz**. After the decompression, you will obtain the following JDBC packages in .jar format:

- **gaussdbjdbc.jar**: The main class name is **com.huawei.gaussdb.jdbc.Driver**. The URL prefix of the database connection is **jdbc:gaussdb**. This driver package is recommended. The Java code examples in this section use the **gaussdbjdbc.jar** package by default.
- **gscejdbc.jar**: The main class name is **com.huawei.gaussdb.jdbc.Driver**. The URL prefix of the database connection is **jdbc:gaussdb**. This driver package contains the dependent libraries related to encryption and decryption that need to be loaded to the encrypted database. You are advised to use this driver package in encrypted scenarios. Currently, only EulerOS is supported.
- **gaussdbjdbc-JRE7.jar**: The main class name is **com.huawei.gaussdb.jdbc.Driver**. The URL prefix of the database connection is **jdbc:gaussdb**. The **gaussdbjdbc-JRE7.jar** package is used in the JDK 1.7 environment.

CAUTION

- Before using the **gscejdbc.jar** driver package, you need to set the environment variable *LD_LIBRARY_PATH*. For details, see section "Setting Encrypted Equality Queries > Using JDBC to Operate an Encrypted Database" in *Feature Guide*.
- In JDK 1.8, you are advised to use **gaussdbjdbc.jar** instead of **gaussdbjdbc-JRE7.jar**.
- For details about other JAR packages, see **JDBC Compatibility Package**.

Driver Class

Before establishing a database connection, load the **com.huawei.gaussdb.jdbc.Driver** database driver class.

NOTE

1. GaussDB is compatible with PostgreSQL in the use of JDBC. Therefore, when two JDBC drivers are used in the same process, class names may conflict.
2. GaussDB JDBC driver has the following enhanced features:
   1. The SHA256 encryption mode is supported for login.
   2. The third-party log framework that implements the sf4j API can be connected.
   3. DR failover is supported.

Environment Class

JDK1.8 must be configured on the client. JDK supports multiple platforms such as Windows and Linux. The following uses Windows as an example to describe how to configure JDK 1.8:

Step 1 Enter **java -version** in the MS-DOS window (command prompt in Windows) to check the JDK version. Ensure that the JDK version is JDK1.8. If JDK is not installed, download the installation package from the official website and install it.

Step 2 Configure system environment variables.

1. Right-click **My computer** and choose **Properties**.
2. In the navigation pane, choose **Advanced system settings**.
3. In the **System Properties** dialog box, click **Environment Variables** on the **Advanced** tab page.
4. In the **System variables** area of the **Environment Variables** dialog box, click **New** or **Edit** to configure system variables. For details about the variables, see **Table 5-3**.

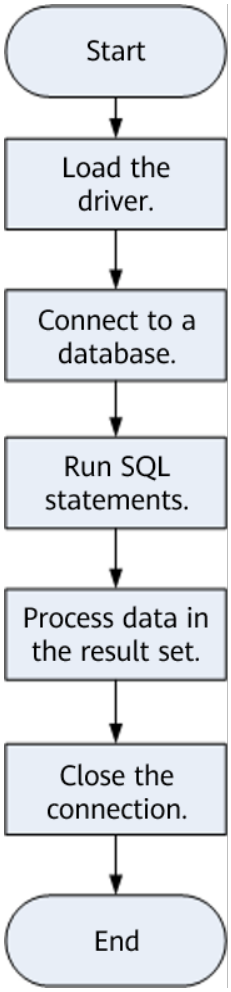
Table 5-3 Variables

Variable	Operation	Variable Value
JAVA_HOME	– If the variable exists, click **Edit**. – If the variable does not exist, click **New**.	Specifies the Java installation directory. Example: **C:\Program Files\Java\jdk1.8.0_131**.
Path	Click **Edit**.	– If *JAVA_HOME* is configured, add *%JAVA_HOME%***\bin** before the variable value. – If *JAVA_HOME* is not configured, add the following full Java installation path before the variable value: **C:\Program Files\Java\jdk1.8.0_131\bin**
CLASSPATH	Click **New**.	**%JAVA_HOME%\lib;%JAVA_HOME%\lib\tools.jar**

----End

5.3.2 Development Process

Figure 5-1 Application development process based on JDBC



5.3.3 Loading the Driver

Load the database driver before creating a database connection.

You can load the driver in the following ways:

- Implicit loading at any position before a connection is created in the code: **Class.forName("com.huawei.gaussdb.jdbc.Driver");**
- Parameter transfer during JVM startup: **java -Djdbc.drivers=com.huawei.gaussdb.jdbc.Driver jdbctest**

  NOTE

  jdbctest is the name of a test application.

5.3.4 Connecting to a Database

After a database is connected, it can be used to run SQL statements to operate data.

Function Prototype

JDBC provides the following three database connection methods:

- DriverManager.getConnection(String url)
- DriverManager.getConnection(String url, Properties info)
- DriverManager.getConnection(String url, String user, String password)

Parameters

Table 5-4 Database connection parameters

Parameter	Description
url	**gaussdbjdbc.jar** database connection descriptor. If **host** is set to a server name or an IPv4 address, formats are as follows: ● jdbc:gaussdb: (If the database name is left empty, the username is used.) ● jdbc:gaussdb:database ● jdbc:gaussdb://host/database ● jdbc:gaussdb://host:port/database ● jdbc:gaussdb://host:port/database?param1=value1&param2=value2 ● jdbc:gaussdb://host1:port1,host2:port2/database?param1=value1&param2=value2 If **host** is set to an IPv6 address, formats are as follows: ● jdbc:gaussdb: (If the database name is left empty, the username is used.) ● jdbc:gaussdb:database ● jdbc:gaussdb://host/database or jdbc:gaussdb://[host]/database ● jdbc:gaussdb://[host]:port/database ● jdbc:gaussdb://[host]:port/database?param1=value1&param2=value2 ● jdbc:gaussdb://[host1]:port1,[host2]:port2/database?param1=value1&param2=value2

Parameter	Description
	NOTE • **database** indicates the name of the database to connect. • **host** indicates the name or IP address of the database server. Both IPv4 and IPv6 addresses are supported. For security purposes, the primary database node forbids access from other nodes in the database without authentication. To access the primary database node from inside the database, deploy the JDBC program on the host where the primary database node is located and set **host** to **127.0.0.1**. Otherwise, the error message "FATAL: Forbid remote connection with trust method!" may be displayed. It is recommended that the service system be deployed outside the database. Otherwise, the database performance may be affected. By default, the local host is used to connect to the server. • **port** indicates the port number of the database server. By default, the database on port 5432 of the local host is connected. • If **host** is set to an IPv6 address and the port number is specified in the URL, use square brackets ([]) to enclose the IP address. The format is [IP address]:Port number. • **param** indicates a database connection attribute. The parameter can be configured in the URL. The URL starts with a question mark (?), uses an equal sign (=) to assign a value to the parameter, and uses an ampersand (&) to separate parameters. You can also use the attributes of the info object for configuration. For details, see **Examples**. • **value** indicates the database connection attribute values. • The **connectTimeout** and **socketTimeout** parameters must be set for connection. If they are not set, the default value **0** is used, indicating that the connection will not time out. When the network between the DN and client is faulty, the client does not receive the ACK packet from the DN. In this case, the client starts the timeout retransmission mechanism to continuously retransmit packets. A timeout error is reported only when the timeout interval reaches the default value **600s**. As a result, the RTO is high. • You are advised to ensure the validity of the URL when using the standard JDBC interface to establish a connection. An invalid URL may cause an exception, and the exception contains the original URL character string, which may cause sensitive information leakage.

Parameter	Description
info	Database connection attributes (all attributes are case sensitive). Common attributes are described as follows: • **PGDBNAME**: string type. It specifies the database name. You do not need to set this parameter in the URL because the database name is automatically parsed from the URL. • **PGHOST**: string type. It specifies the host IP address. Both IPv4 and IPv6 addresses are supported. For details, see **Examples**. • **PGPORT**: integer type. It specifies the host port number. For details, see **Examples**. • **user**: string type. It specifies the database user who creates the connection. • **password**: string type. It specifies the password of the database user. • **enable_ce**: string type. **enable_ce=1** indicates that JDBC supports the basic capability of encrypted equality query. **enable_ce=3** indicates that software and hardware integration is supported based on the encrypted equality query capability. • **key_info**: string type. This parameter is used together with **enable_ce** to set parameters for accessing an external key manager in an encrypted database. • **refreshClientEncryption**: string type. The default value is **NULL**. If **refreshClientEncryption** is set to **1** (default value), the encrypted database supports cache update on the client. • **loggerLevel**: string type. The default value is **NULL** (disabled). The following log levels are supported: **OFF**, **INFO**, **DEBUG**, and **TRACE**. **OFF** indicates that the log function is disabled. **INFO**, **DEBUG**, and **TRACE** logs record information of different levels. • **loggerFile**: string type. It specifies the log output path (directory and file name). You need to specify the log directory and file name. If no directory is specified, log files are generated in the directory where the client program is running. This parameter has been discarded and does not take effect any more. It is used to specify the path for exporting monitoring logs only when the connection monitoring function is enabled. To export JDBC logs to a specified path, configure it in the **java.util.logging** attribute file or system attributes. • **logger**: string type. It indicates the log output framework used by the JDBC driver. The JDBC driver supports the log output framework used for interconnecting with user applications. Currently, only the Slf4j-API-based third-party log output framework is supported. For details, see **Log Management**. If it is left empty or is set to **JDK LOGGER**, JDK LOGGER is used. Otherwise, the slf4j-API-based third-party log framework must be used. • **allowEncodingChanges**: Boolean type. The default value is **false**. If this parameter is set to **true**, the character set type can be

Parameter	Description
	changed. This parameter is used together with **characterEncoding** to set the character set. The two parameters are separated by ampersands (&). The value of **characterEncoding** can be **UTF8**, **GBK**, **LATIN1**, **GB18030**, **GB18030_2022**, or **ZHS16GBK**. Example: **allowEncodingChanges=true&characterEncoding=UTF8**. **NOTE** The ZHS16GBK character set supports the euro sign. To display the euro sign, set the character set to **ZHS16GBK**. • **currentSchema**: string type. You need to specify a schema in **search-path**. If the schema name contains special characters except letters, digits, and underscores (_), you are advised to enclose the schema name in quotation marks. Note that the schema name is case sensitive after quotation marks are added. If multiple schemas need to be configured, separate them with commas (,). Schemas containing special characters also need to be enclosed in quotation marks. Example: **currentSchema=schema_a,"schema-b","schema/c"**. • **hostRecheckSeconds**: integer type. After JDBC attempts to connect to a host, the host status is saved: connection success or connection failure. This status is trusted within the duration specified by **hostRecheckSeconds**. After the duration expires, the status becomes invalid. The default value is 10 seconds. • **ssl**: Boolean type. It specifies a connection in SSL mode. When **ssl** is set to **true**, the NonValidatingFactory channel and certificate mode are supported. 1. For the NonValidatingFactory channel, configure the username and password and set **SSL** to **true**. 2. In certification mode, configure the client certificate, key, and root certificate, and set **SSL** to **true**. • **sslmode**: string type. It specifies the SSL authentication mode. The value can be **disable**, **allow**, **prefer**, **require**, **verify-ca**, or **verify-full**. – **disable**: SSL connection is disabled. – **allow**: If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified. – **prefer**: If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified. – **require**: The system attempts to set up an SSL connection. If there is a CA file, the system performs verification as if the parameter was set to **verify-ca**. – **verify-ca**: attempts to set up an SSL connection and checks whether the server certificate is issued by a trusted CA. – **verify-full**: The system attempts to set up an SSL connection, checks whether the server certificate is issued by a trusted CA,

Parameter	Description
	and checks whether the host name of the server is the same as that in the certificate. • **sslcert**: string type. It specifies the complete path of the certificate file. The type of the client and server certificates is **End Entity**. • **sslkey**: string type. It specifies the complete path of the key file. Convert the client certificate to the DER format. `openssl pkcs8 -topk8 -outform DER -in client.key -out client.key.pk8 -nocrypt` • **sslrootcert**: string type. It specifies the name of the SSL root certificate. The root certificate type is CA. • **sslpassword**: string type. It is provided for ConsoleCallbackHandler. • **sslpasswordcallback**: string type. It specifies the class name of the SSL password provider. Default value: **com.huawei.gaussdb.jdbc.ssl.jdbc4.LibPQFactory.ConsoleCallbackHandler**. • **sslfactory**: string type. It specifies the class name used by SSLSocketFactory to establish an SSL connection. • **sslprivatekeyfactory**: string type. The provided value is the fully qualified name of the implementation class of the com.huawei.gaussdb.jdbc.ssl.PrivateKeyFactory interface that implements the private key decryption method. If it is not provided, try the default JDK private key decryption algorithm. If the decryption fails, use com.huawei.gaussdb.jdbc.ssl.BouncyCastlePrivateKeyFactory. You need to provide the **bcpkix-jdk15on.jar** package. The recommended version is 1.65 or later. • **sslfactoryarg**: string type. The value is an optional parameter of the constructor function of the sslfactory class. (This parameter is not recommended.) • **sslhostnameverifier**: string type. It specifies the class name of the host name verifier. The interface must implement javax.net.ssl.HostnameVerifier. The default value is **com.huawei.gaussdb.jdbc.ssl.PGjdbcHostnameVerifier**. • **loginTimeout**: integer type. It specifies the waiting time for establishing the database connection, in seconds. When multiple IP addresses are configured in the URL, if the time for obtaining the connection exceeds this value, the connection fails and the subsequent IP addresses are not tried. The default value is **0**. • **connectTimeout**: integer type. It specifies the timeout interval for connecting to a server. If the time taken to connect to a server exceeds the value specified, the connection is interrupted. The unit of the timeout interval is second. The value **0** indicates that the timeout mechanism does not take effect. When multiple IP addresses are configured in the URL, it indicates the timeout interval for connecting to a single IP address. The default value is **0**.

Parameter	Description
	• **socketTimeout**: integer type. It specifies the timeout interval for a socket read operation. If the time taken to read data from a server exceeds the value specified, the connection is closed. The unit of the timeout interval is second. The value **0** indicates that the timeout mechanism does not take effect. The default value is **0**. If this parameter is not set, the client waits for a long time when the database process is abnormal. You are advised to set this parameter based on the SQL execution time acceptable to services. When the timeout is triggered on the JDBC and the connection is closed, the running services delivered by the JDBC to the database are forcibly terminated. This capability is controlled by the GUC parameter **check_disconnect_query**. If this parameter is set to **on**, the capability is supported. If this parameter is set to **off**, the capability is not supported. • **socketTimeoutInConnecting**: integer type. It specifies the timeout interval for a socket read operation during the connection establishment. If the time taken to read data from the server exceeds this value, it searches for the next node for connection. The unit of the timeout interval is second. The default value is **5s**. • **statementTimeout**: integer type. It specifies the timeout interval for executing a statement in a connection. If the execution time of a statement exceeds this value, the statement execution is canceled. The unit of the timeout interval is millisecond. The default value **0** indicates that the timeout mechanism is disabled. • **cancelSignalTimeout**: integer type. A cancel message may cause a block. This attribute controls connect timeout and socket timeout in a cancel command. If the cancel command does not respond within the specified time, the connection is interrupted to reduce the occupation of client resources. The default value is 10 seconds. • **tcpKeepAlive**: Boolean type. It is used to enable or disable TCP keepalive detection. The default value is **false**. • **logUnclosedConnections**: Boolean type. The default value is **false**. The client may leak a connection object because it does not call the connection object's close() method. These objects will be collected as garbage and finalized using the finalize() method. After it is set to **true**, if the caller ignores this operation, this method closes the connection. • **assumeMinServerVersion** (discarded): string type. The client sends a request to set a floating point. This parameter specifies the version of the server to connect, for example, **assumeMinServerVersion=9.0**. This parameter can reduce the number of packets to send during connection setup. • **ApplicationName**: string type. It specifies the name of the JDBC driver that is being connected. You can query the **pg_stat_activity** table on the primary database node to view information about the

Parameter	Description
	client that is being connected. The JDBC driver name is displayed in the **application_name** column. The default value is **GaussDB JDBC Driver**. • **connectionExtraInfo**: Boolean type. It specifies whether the driver reports the driver deployment path, process owner, and URL connection configuration information to the database. The value can be **true** or **false**. The default value is **false**. If **connectionExtraInfo** is set to **true**, the JDBC driver reports the driver deployment path, process owner, and URL connection configuration information to the database and displays the information in the **connection_info** parameter. In this case, you can query the information from PG_STAT_ACTIVITY. • **autosave**: string type. The value can be **always**, **never**, or **conservative**. The default value is **never**. It specifies the action that the driver should perform upon a query failure. If **autosave** is set to **always**, the JDBC driver sets a savepoint before each query and rolls back to the savepoint if the query fails. If **autosave** is set to **never**, there is no savepoint. If **autosave** is set to **conservative**, a savepoint is set for each query. However, the system rolls back and retries only when there is an invalid statement. The default value is **never**. • **protocolVersion**: integer type. It specifies the connection protocol version. Only versions 1 and 3 are supported. If it is set to **1**, only the V1 server is connected. If it is set to **3**, MD5 encryption is used. You must need to set the GUC parameter **password_encryption_type** to **1** to change the database encryption mode. After the database is restarted, create a user who uses MD5 encryption to encrypt passwords. In addition, modify the **gs_hba.conf** file to change the client connection mode to MD5. Log in as a new user. (You are advised not to set this parameter because the MD5 encryption algorithm has lower security and poses security risks.) **NOTE** The MD5 encryption algorithm has lower security and poses security risks. Therefore, you are advised to use a more secure encryption algorithm. • **prepareThreshold**: integer type. It specifies the number of times that the PreparedStatement object is executed before the prepared statement on the server is used. The default value is **5**, indicating that when the same PreparedStatement object is executed for five or more times, the parse message is not sent to the server to parse the statement. Instead, the statement that has been parsed on the server is used. • **preparedStatementCacheQueries**: integer type. It specifies the number of queries cached in each connection. The default value is **256**. If more than 256 different queries are used in the prepareStatement() call, the least recently used query cache will be discarded. The value **0** indicates that the cache function is disabled.

Parameter	Description
	• **preparedStatementCacheSizeMiB**: integer type. This parameter indicates the maximum size of queries generated by the cache statement object of each connection, in MB. The default value is **5**. If the size of the cached queries exceeds 5 MB, the least recently used query cache will be discarded. The value **0** indicates that the cache function is disabled. • **databaseMetadataCacheFields**: integer type. The default value is **65536**. It specifies the maximum number of columns that can be cached in each connection. The value **0** indicates that the cache function is disabled. • **databaseMetadataCacheFieldsMiB**: integer type. The default value is **5**. It specifies the maximum size of columns that can be cached for each connection, in MB. The value **0** indicates that the cache function is disabled. • **stringtype**: string type. The options are **unspecified** and **varchar**. It specifies the type of the **PreparedStatement** parameter used by the setString() method. If **stringtype** is set to **VARCHAR** (default value), these parameters are sent to the server as varchar parameters. If **stringtype** is set to **unspecified**, these parameters are sent to the server as an untyped value, and the server attempts to infer their appropriate type. • **batchMode**: string type. It specifies whether to connect the database in batch mode. The default value is **on**, indicating that the batch mode is enabled. If **batchMode** is set to **on**, the returned result is [count, 0, 0...0]. The first element in the array is the total number of records affected in batches. If **batchMode** is set to **off**, the returned result is [1, 1, 1...1]. Each element in the array corresponds to the number of affected records in a single modification. • **fetchsize**: integer type. It specifies the default **fetchsize** for statements in the created connection. The default value is **0**, indicating that all results are obtained at a time. It has the same function as **defaultRowFetchSize**. If they are set at the same time, **fetchsize** prevails. • **reWriteBatchedInserts**: Boolean type. The default value is **false**. During batch import, set this parameter to **true** to combine *N* insert statements into one: insert into TABLE_NAME values(values1, ..., valuesN), ..., (values1, ..., valuesN). To use this parameter, set **batchMode** to **off**. • **unknownLength**: integer type. The default value is **Integer.MAX_VALUE**. This parameter specifies the length of the unknown length type when the data of some GaussDB types (such as TEXT) is returned by functions such as ResultSetMetaData.getColumnDisplaySize and ResultSetMetaData.getPrecision. • **uppercaseAttributeName**: Boolean type. The default value is **false** (disabled). The value can also be **true** (enabled). If this parameter is enabled, the query result of the API for obtaining

Parameter	Description
	metadata is converted to uppercase letters. The application scenario is as follows: All metadata stored in the database is in lowercase, but the metadata in uppercase must be used as the input and output parameters. For details about the involved APIs, see **java.sql.DatabaseMetaData** and **java.sql.ResultSetMetaData**. • **defaultRowFetchSize**: integer type. It specifies the number of rows read by fetch in ResultSet at a time. Limiting the number of rows read each time in a database access request can avoid unnecessary memory consumption, thereby avoiding out of memory exception. The default value is **0**, indicating that all rows are obtained at a time in ResultSet. This parameter cannot be set to a negative value. • **binaryTransfer**: Boolean type. It specifies whether data is sent and received in binary format. The default value is **false**. • **binaryTransferEnable**: string type. It specifies types for which binary transmission is enabled. Every two types are separated by commas (,). You can select either the OID or name, for example, binaryTransferEnable=Integer4_ARRAY,Integer8_ARRAY. For example, if the name is BLOB and the OID is 88, set **binaryTransferEnable** to **BLOB** or **88**. • **binaryTransferDisable**: string type. It specifies types for which binary transmission is disabled. Every two types are separated by commas (,). You can select either the OID or name. It overwrites the setting of **binaryTransferEnable**. • **blobMode**: string type. It sets the setBinaryStream method to assign values to different types of data. The value **on** indicates that values are assigned to BLOB data. The value **off** indicates that values are assigned to bytea data. The default value is **on**. You are advised to set it to **on** for systems migrated from Oracle or MySQL and to **off** for systems migrated from PostgreSQL. • **socketFactory**: string type. It specifies the name of the class used to create a socket connection with the server. This class must implement the javax.net.SocketFactory API and define a constructor with no parameter or a single string parameter. • **socketFactoryArg**: string type. The value is an optional parameter of the constructor function of the socketFactory class and is not recommended. • **receiveBufferSize**: integer type. It is used to set **SO_RCVBUF** on the connection stream. • **sendBufferSize**: integer type. It is used to set **SO_SNDBUF** on the connection stream. • **preferQueryMode**: string type. The value can be **"extended"**, **"extendedForPrepared"**, **"extendedCacheEverything"**, or **"simple"**. It specifies the query mode. In **simple** mode, the query is executed without parsing or binding. In **extended** mode, the

Parameter	Description
	query is executed and bound. The **extendedForPrepared** mode is used for prepared statement extension. In **extendedCacheEverything** mode, each statement is cached. • **targetServerType**: string type. This parameter is used to identify the primary DN and standby DN by querying whether a DN allows the write operation in the URL connection string. The default value is **any**. The value can be **"any"**, **"master"**, **"slave"**, **"preferSlave"**, or **"clusterMainNode"**. – **master**: attempts to connect to a primary node in the URL connection string. If the primary node cannot be found, an exception is thrown. – **slave**: attempts to connect to a standby node in the URL connection string. If the primary node cannot be found, an exception is thrown. – **preferSlave**: attempts to connect to a standby DN (if available) in the URL connection string. Otherwise, it connects to the primary DN. – **any**: attempts to connect to any DN in the URL connection string. – **clusterMainNode**: attempts to connect to the primary node or main standby node (primary DR node) in the URL string. If the primary node or main standby node cannot be found, an exception is thrown. • **priorityServers**: integer type. This value is used to specify the first *n* nodes configured in the URL as the primary database instance to be connected preferentially. The default value is **NULL**. The value is a number greater than 0 and less than the number of DNs configured in the URL. It is used in streaming DR scenarios. Example: **jdbc:gaussdb://host1:port1,host2:port2,host3:port3,host4:port4,/database?priorityServers=2**. That is, **host1** and **host2** are primary database instance nodes, and **host3** and **host4** are DR database instance nodes. • **forceTargetServerSlave**: Boolean type. It specifies whether to enable the function of forcibly connecting to the standby node and forbid the existing connections to be used on the standby node that is promoted to primary during the primary/standby switchover of the database instance. The default value is **false**, indicating that the function of forcibly connecting to the standby node is disabled. **true**: The function of forcibly connecting to the standby node is enabled. • **traceInterfaceClass**: string type. The default value is **NULL**, which is used to obtain the implementation class of **traceId**. The value is the fully qualified name of the implementation class of the com.huawei.gaussdb.jdbc.log.Tracer API that implements the method for obtaining **traceId**.

Parameter	Description
	• **use_boolean**: Boolean type. It is used to set the OID type bound to the setBoolean method in extended mode. The default value is **false**, indicating that the int2 type is bound. The value **true** indicates that the Boolean type is bound. • **allowReadOnly**: Boolean type. It specifies whether to enable the read-only mode for a connection. The default value is **true**, indicating that the read-only mode is enabled. If it is set to **false**, the read-only mode is disabled. In this case, invoking connection.setReadOnly(true) does not take effect, and data can still be modified. • **TLSCiphersSupperted**: string type. It is used to set the supported TLS encryption suite. The default value is **TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384**. • **stripTrailingZeros**: Boolean type. The default value is **false**. If the value is **true**, trailing 0s of the numeric type are removed. It is valid only for **ResultSet.getObject(int columnIndex)**. • **enableTimeZone**: Boolean type. The default value is **true**. It specifies whether to enable the time zone setting on the server. The value **true** indicates that the JVM time zone is obtained to specify the database time zone. The value **false** indicates that the database time zone is used. • **loadBalanceHosts**: Boolean type. In the default mode (disabled), multiple hosts specified in the URL are connected in default sequence. If load balancing is enabled, the shuffle algorithm is used to randomly select a host from the candidate hosts to establish a connection. In a centralized environment, ensure that no write operation is in services when using this parameter. • **compatibilityTags**: string type. The default value is **NULL**. It is used to restore some features of the driver to an earlier version to ensure forward compatibility. You can configure one or more tags. Each tag indicates that the corresponding forward compatibility feature is enabled. Different tags are separated by commas (,). Generally, this parameter is not recommended unless there is an obvious forward compatibility issue. Currently, the following tag is supported: **typeMapInitNull**: After this tag is used, the behavior of earlier driver versions is compatible. The initial value of **typeMap** in **java.sql.Connection** is **NULL**. If this tag is not used, the initial value is an empty map. • **parseCandidatesByDomain**: Boolean type. Specifies whether to obtain standby nodes based on domain names. The default value is **false**. The value can be **true** (enabled) or **false** (disabled). After this function is enabled, you need to configure the host

Parameter	Description
	information in the URL in the format of domain name plus port number. The driver obtains the IP address based on the domain name, generates a standby node set, and works with the **autoBalance** parameter for load balancing or with the **targetServerType** parameter for automatic leader election. • **primaryDomains**: integer type. It specifies the first *n* domain names configured in the URL as the primary cluster to be preferentially connected. The default value is **0** (disabled). It takes effect only when **parseCandidatesByDomain** is set to **true**. A value greater than **0** and less than the number of domain names configured in the URL indicates that the function is enabled. The domain names in the URL are divided into two groups. The first group is the primary database instance, and the second group is the standby database instance. After a primary/standby switchover, the two groups exchange their positions. The first group is the standby database instance, and the second group is the primary database instance. It applies to streaming DR and Dorado dual-cluster scenarios. Example: **jdbc:gaussdb://domain1:port1,domain2:port2,domain3:port3,domain4:port4/database?parseCandidatesByDomain=true&primaryDomains=2**, in which **domain1** and **domain2** correspond to the primary database instance and the IP addresses corresponding to **domain1** and **domain2** are preferentially connected. If a primary/standby switchover occurs, **domain3** and **domain4** are marked as the primary database instance, and connections are preferentially established to **domain3** and **domain4**. • **priorityDomains**: integer type. It specifies the first *n* domain names configured in the URL as the domain names to be preferentially connected. The default value is **0** (disabled). The function takes effect only when **parseCandidatesByDomain** is set to **true**. A value greater than **0** and less than the number of domain names configured in the URL indicates that the function is enabled. If **primaryDomains** is configured, the value must be less than the value of **primaryDomains**. Example: **jdbc:gaussdb://domain1:port1,domain2:port2,domain3:port3,domain4:port4/database?parseCandidatesByDomain=true&primaryDomains=2&priorityDomains=1**, in which **domain1** and **domain2** correspond to the primary database instance but the IP address corresponding to **domain1** is preferentially connected. **domain2** is the standby domain name. The system attempts to connect to **domain2** only when **domain1** cannot be connected. If a primary/standby switchover occurs, **domain3** and **domain4** are marked as the primary database instance. However, the IP address corresponding to **domain3** is preferentially connected. If **domain3** cannot be connected, the IP address corresponding to **domain4** is connected.

Parameter	Description
	• **refreshDomainResolveTime**: integer type. The default value is **10**, in seconds. The minimum value is **1** and the maximum value is **2147483647**. It specifies the interval for updating the domain name resolution result and takes effect only when **parseCandidatesByDomain** is set to **true**. After it takes effect, the domain name resolution update time involved in the URL is set to a specified value and the domain name resolution result is periodically updated in the asynchronous thread. Driver obtains domain name resolution results only from the DNS service in the environment where the application is located. To ensure that domain name binding changes can be detected by Driver in a timely manner, the DNS service in the environment where the application is located must ensure that the domain name binding changes take effect in a timely manner. • **oracleCompatible**: string type. The default value is **false**. This is used to set the A-compatible features of driver APIs. The options are as follows: 1. **true** or **on**: All A-compatible features of drivers are enabled. 2. **false** or **off**: All A-compatible features of drivers are disabled. 3. **"tag1,tag2,tag3"**: Some A-compatible features of drivers are enabled. You can configure one or more tags separated by commas (,). Each tag corresponds to an A-compatible feature. Currently, the following tags are supported: – getProcedureColumns: The behavior of the DatabaseMetaData#getProcedureColumns API is compatible with behavior A. – getCallableStatementResults: The database is compatible with the A database when the getLong, getInt, getShort, and getByte interfaces of CallableStatement are invoked. When the registered output parameter type is java.sql.Types#NUMERIC, you can invoke the getLong, getInt, getShort, and getByte interfaces of CallableStatement to receive the value of **out**. The SQLException message is displayed only when the value of **out** exceeds the value range of the Java numeric data type. – batchInsertAffectedRows: After reWriteBatchedInserts is enabled, the result returned by the Statement#executeBatch API is compatible with behavior A. • **printSqlInLog**: Boolean type. It specifies whether to output SQL statements in exception information or logs. The value is **true** (enabled) or **false** (disabled). The default value is **true**. • **useGsClobBlobClass**: Boolean type. The default value is **false** (disabled). – If it is set to **true**, an object of the PGClob type is returned when the java.sql.ResultSet#getObject interface is used to

Parameter	Description
	obtain the CLOB columns, and an object of the PGBlob type is returned when the java.sql.ResultSet#getObject interface is used to obtain the BLOB columns. When the metadata interface java.sql.ResultSetMetaData#getColumnClassName is used to obtain the type name of a CLOB column, java.sql.Clob is returned. When the metadata interface java.sql.ResultSetMetaData#getColumnClassName is used to obtain the type name of a BLOB column, java.sql.Blob is returned. – If it is set to **false**, an object of the PGClob type is returned when the java.sql.ResultSet#getObject interface is used to obtain the CLOB columns, and an object of the byte[] type is returned when the java.sql.ResultSet#getObject interface is used to obtain the BLOB columns. If the type name of the CLOB column is obtained through the metadata API java.sql.ResultSetMetaData#getColumnClassName, java.sql.CLOB is returned. If the type name of the BLOB column is obtained, java.sql.BLOB is returned. • **executeUpdateQueryable**: Boolean type. It specifies whether to enable the executeUpdate method to execute DQL statements. The value is **true** (enabled) or **false** (disabled). The default value is **false**. After this method is enabled, **–1** is returned when the executeUpdate method executes DQL statements, and a result set can be obtained. You are advised not to enable this parameter. • **dbMonitor**: Boolean type. It specifies whether to enable the connection monitoring function for JDBC. The default value is **false**. The value can be **true** (enabled) or **false** (disabled). – The connection monitoring function can monitor the following JDBC metrics: number of connections enabled by applications, number of connections disabled by applications, number of abnormal disconnections, database access volume, client CPU usage, memory usage, uplink and downlink transmission rates, and network delay, jitter, and packet loss rate between applications and databases. – If **dbMonitor** is set to **true**, **loggerLevel** is set to **debug**, and **loggerFile** is set to **filePath**, the client connection monitoring information is recorded in the **filePath** log file. For details, see **Example 4: Using the Database Connection Monitoring Function**. • **enableStreamingQuery**: Boolean type. It specifies whether to enable the streaming read function. The default value is **false**. The value can be **true** (enabled) or **false** (disabled). If this parameter is set to **true** and statement.setFetchSize(Integer.MIN_VALUE) or statement.enableStreamingResults() is used, the streaming read function is enabled. Streaming read: All data is read at a time and sent to the socket buffer of a client until the buffer is full. If there is free space, the

Parameter	Description
	data continues to be sent to the buffer. At the same time, a JVM reads data from the buffer row by row. The advantage is that the result is processed fast and no JVM memory overflow occurs. The disadvantage is that only backward traversal is supported. Before the data processing is complete or the statement is closed, other operations cannot be performed on the current connection. For details, see example 2 in **Examples**. • **yearIsDateType**: Boolean type. The default value is **true** (enabled). – **true**: An object of the date type is returned when the java.sql.ResultSet#getObject interface is used to obtain columns of the year type. A date character string in the yyyy-mm-dd format is returned when the java.sql.ResultSet#getString interface is used to obtain data. A year value is returned when the java.sql.ResultSet#getInt/ getLong/getShort interface is used to obtain data. – **false**: An object of the integer type is returned when the java.sql.ResultSet#getObject interface is used to obtain columns of the year data type. A year character string is returned when the java.sql.ResultSet#getString interface is used to obtain data. A year value is returned when the java.sql.ResultSet#getInt/getLong/getShort interface is used to obtain data. • **enableALT**: Boolean type. The default value is **false** (disabled). To enable an ALT function, you must configure both **enableALT** and **gns**. – **true**: When a planned ALT function is enabled to connect to a database, the database instance first connects to GNS. The database instance will not reconnect to GNS when connecting to DNs. When the database instance status changes, GNS sends a FAN message to JDBC. – **false**: The planned ALT function is disabled, and JDBC does not connect to GNS. • **altLevel**: Char type. The default value is empty, indicating that only the database instance status notification is enabled for ALT. – If it is set to **C**, the connection acceleration and quick disconnection functions are enabled. When a new connection is established to a DN, the DN list is sorted based on the primary and standby status of each node in the database instance connection manager to accelerate the establishment of the new connection. After receiving a notification indicating that the database instance node status is **DOWN**, the database instance connection manager automatically closes connections managed by itself. – If it is set to **P**, the connection acceleration, quick disconnection, and planned maintenance functions are enabled.

Parameter	Description
	When a new connection is established to a DN, the DN list is sorted based on the primary and standby status of each node in the database instance connection manager to accelerate the establishment of the new connection. After receiving a notification indicating that the database instance node status is **DOWN**, the database instance connection manager automatically closes connections managed by itself. After receiving a message indicating that the database instance node enters the planned maintenance state, the transaction drain is performed for all connections in the database instance connection manager. After the planned maintenance is complete, DNs are reconnected and GUC parameters of the connections are restored to ensure that the connections are available. • **gns**: string type. The default value is empty. It specifies a list of IP addresses and ports of GNSs. JDBC connects to GNSs in the GNS list in a random sequence until a connection is established, fails, or times out. • **setFloat**: Boolean type, indicating whether the OID transferred to the kernel is the OID of float4 when setFloat is called or **setObject** is set to **float**. The value **true** (enabled) indicates that the OID transferred to the kernel is float8. The value **false** (disabled) indicates that the OID transferred to the kernel is float4. The default value is **false**.
user	Database user.
password	Password of the database user.

NOTE

After the **uppercaseAttributeName** parameter is enabled, if the database contains metadata with a mixture of uppercase and lowercase letters, only the metadata in lowercase letters can be queried and output in uppercase letters. Before using the metadata, ensure that the metadata is stored in lowercase letters to prevent data errors.

Examples

Example 1: Connect to a database.

```
// The following code encapsulates database connection operations into an interface. The database can then be connected using an authorized username and a password.
public static Connection getConnect(String username, String passwd)
   {
      // Driver class.
      String driver = "com.huawei.gaussdb.jdbc.Driver";
      // Database connection descriptor.
      String sourceURL = "jdbc:gaussdb://$ip:$port/database";
      Connection conn = null;

      try
```

```
    {
        // Load the driver.
        Class.forName(driver);
    }
    catch( Exception e )
    {
        e.printStackTrace();
        return null;
    }

    try
    {
         // Create a connection.
        conn = DriverManager.getConnection(sourceURL, username, passwd);
        System.out.println("Connection succeed!");
    }
    catch(Exception e)
    {
        e.printStackTrace();
        return null;
    }

    return conn;
}
```

Example 2: Use the Properties object as a parameter to create a connection.

```
// The following code uses the Properties object as a parameter to establish a connection:
public static Connection getConnectUseProp(String username, String passwd)
{
    // Driver class.
    String driver = "com.huawei.gaussdb.jdbc.Driver";
    // Database connection descriptor.
    String sourceURL = "jdbc:gaussdb://$ip:$port/database?";
    Connection conn = null;
    Properties info = new Properties();

    try
    {
        // Load the driver.
        Class.forName(driver);
    }
    catch( Exception e )
    {
        e.printStackTrace();
        return null;
    }

    try
    {
         info.setProperty("user", username);
         info.setProperty("password", passwd);
         // Create a connection.
         conn = DriverManager.getConnection(sourceURL, info);
         System.out.println("Connection succeed!");
    }
    catch(Exception e)
    {
        e.printStackTrace();
        return null;
    }

    return conn;
}
```

For details about common parameters, see **Common JDBC Parameters**.

Example 3: Use the streaming read function.


```
// There will be security risks if the username and password used for authentication are directly written into code. It is recommended that the username and password be stored in the configuration file or environment variables (the password must be stored in ciphertext and decrypted when being used) to ensure security.
// In this example, the username and password are stored in environment variables. Before running this example, set environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in the local environment (set the environment variable names based on the actual situation).
// Establish a connection.
public static Connection getConnection(String username, String passwd) {
    String driver = "com.huawei.gaussdb.jdbc.Driver";
    String sourceURL = "jdbc:gaussdb://$ip:$port/database?enableStreamingQuery=true";
    Connection conn = null;
    try {
            // Load the driver.
        Class.forName(driver);
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
    try {
             // Create a connection.
        conn = DriverManager.getConnection(sourceURL, username, passwd);
        System.out.println("Connection succeed!");
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
    return conn;
}

// Execute common SQL statements to create table t_user.
public static void CreateTable(Connection conn) {
    Statement stmt = null;
    try {
        stmt = conn.createStatement();

       // Execute common SQL statements.
        stmt.executeUpdate("DROP TABLE IF EXISTS t_user");
        stmt.executeUpdate("CREATE TABLE t_user(id int, name VARCHAR(20));");
        stmt.close();
    } catch (SQLException e) {
        if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
        e.printStackTrace();
    }
}

// Execute a prepared statement to insert data in batches.
public static void BatchInsertData(Connection conn) {
    PreparedStatement pst = null;

    try {
        // Generate a prepared statement.
        pst = conn.prepareStatement("INSERT INTO t_user VALUES (?,?)");
        for (int i = 0; i < 20; i++) {
            // Add parameters.
            pst.setInt(1, i + 1);
            pst.setString(2, "name " + (i + 1));
            pst.addBatch();
        }
        // Perform batch processing.
        pst.executeBatch();
        pst.close();
    } catch (SQLException e) {
```

```
        if (pst != null) {
            try {
                pst.close();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
        e.printStackTrace();
    }
}

// Enable streaming read and query the content in the t_user table.
public static void StreamingQuery(Connection conn) {
    PreparedStatement pst = null;
    ResultSet resultSet = null;

    try {
        // Query all values in the t_user table.
        pst = conn.prepareStatement("SELECT * FROM t_user");
        pst.setFetchSize(Integer.MIN_VALUE);// Functions the same as
((PgStatement)statement).enableStreamingResults();.
        resultSet = pst.executeQuery();
        while (resultSet.next()) {
            System.out.println(resultSet.getInt(1));
        }
    } catch (SQLException e) {
        throw new RuntimeException(e);
    } finally {
        if (resultSet != null) {
            try {
                resultSet.close();
            } catch (SQLException e) {
                throw new RuntimeException(e);
            }
        }

        if (pst != null) {
            try {
                pst.close();
            } catch (SQLException e) {
                throw new RuntimeException(e);
            }
        }
    }
}

public static void main(String[] args) throws Exception {
    String userName = System.getenv("EXAMPLE_USERNAME_ENV");
    String password = System.getenv("EXAMPLE_PASSWORD_ENV");

    Connection conn = getConnection(userName, password);

    CreateTable(conn);

    BatchInsertData(conn);

    StreamingQuery(conn);

    // Close the connection to the database.
    try {
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```


NOTICE

When the streaming read function is used, you need to perform the resultSet.close() or statement.close() operation after the result set is used. Otherwise, the current connection is unavailable.

5.3.5 Connecting to a Database (Using SSL)

When establishing connections to the GaussDB server using JDBC, you can enable SSL connections to encrypt client and server communications for security of sensitive data transmission on the Internet.

This section describes how applications configure the client in SSL mode through the JDBC driver. For details about how to configure the server, contact the administrator.

To use the method described in this section, you must have the server certificate, client certificate, and private key files. For details on how to obtain these files, see related documents and commands of OpenSSL. For details about how to generate a certificate, contact the administrator.

Configuring the Client

Upload the certificate files **client.key.pk8**, **client.crt**, and **cacert.pem** generated in section "Configuring the Server" to the client.

Examples

Note: Select either example 1 or example 2.

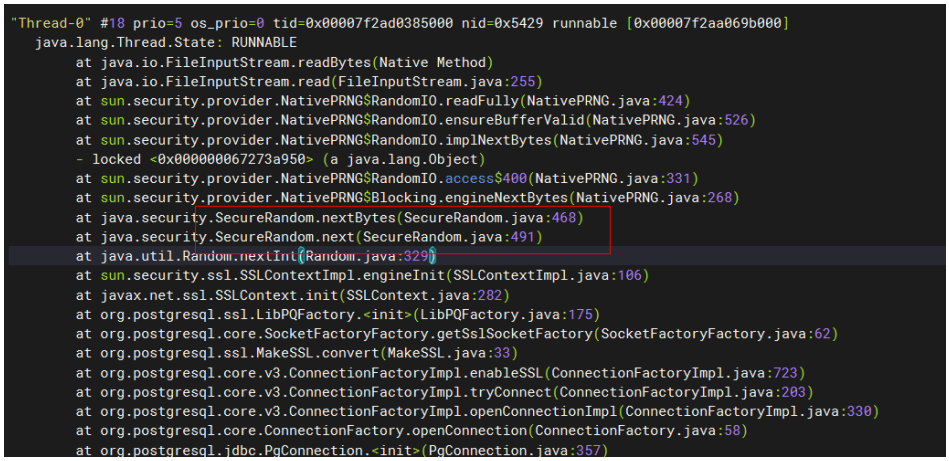
```
// There will be security risks if the username and password used for authentication are directly written into code. It is recommended that the username and password be stored in the configuration file or environment variables (the password must be stored in ciphertext and decrypted when being used) to ensure security.
// In this example, the username and password are stored in environment variables. Before running this example, set environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in the local environment (set the environment variable names based on the actual situation).
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.Properties;

public class SSL{
    public static void main(String[] args) {
        Properties urlProps = new Properties();
        String urls = "jdbc:gaussdb://$ip:$port/postgres";
        String userName = System.getenv("EXAMPLE_USERNAME_ENV");
        String password = System.getenv("EXAMPLE_PASSWORD_ENV");

        /**
         * ================== Example 1: Use the NonValidatingFactory channel.
         */
        urlProps.setProperty("sslfactory","com.huawei.gaussdb.jdbc.ssl.NonValidatingFactory");
        urlProps.setProperty("user", userName);
        urlProps.setProperty("password", password);
        urlProps.setProperty("ssl", "true");
        /**
         * ==================  Examples 2: Use a certificate.
         */
        urlProps.setProperty("sslcert", "client.crt");
        urlProps.setProperty("sslkey", "client.key.pk8");
```



```
        urlProps.setProperty("sslrootcert", "cacert.pem");
        urlProps.setProperty("user", userName);
        urlProps.setProperty("password", password);
        urlProps.setProperty("ssl", "true");
        /* sslmode can be set to require, verify-ca, or verify-full. Select one from the following three
examples: */
       /* ================== Example 2.1: Set sslmode to require to use the certificate for authentication.
*/
        urlProps.setProperty("sslmode", "require");
       /* ================== Example 2.2: Set sslmode to verify-ca to use the certificate for
authentication. */
        urlProps.setProperty("sslmode", "verify-ca");
       /* ================== Example 2.3: Set sslmode to verify-full to use the certificate (in the Linux
OS) for authentication. */
        urls = "jdbc:gaussdb://world:8000/postgres";
        urlProps.setProperty("sslmode", "verify-full");
        try {
            Class.forName("com.huawei.gaussdb.jdbc.Driver").newInstance();
        } catch (Exception e) {
            e.printStackTrace();
        }
        try {
            Connection conn;
            conn = DriverManager.getConnection(urls,urlProps);
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
/**
 * Note: Convert the client key to the DER format.
 * openssl pkcs8 -topk8 -outform DER -in client.key -out client.key.pk8 -nocrypt
 * openssl pkcs8 -topk8 -inform PEM -in client.key -outform DER -out client.key.der -v1 PBE-MD5-DES
 * openssl pkcs8 -topk8 -inform PEM -in client.key -outform DER -out client.key.der -v1 PBE-SHA1-3DES
 * The preceding algorithms are not recommended due to their low security.
 * If the customer needs to use a higher-level private key encryption algorithm, the following private key
encryption algorithms can be used after the BouncyCastle or a third-party private key is used to decrypt the
password package:
 * openssl pkcs8 -in client.key -topk8  -outform DER -out client.key.der -v2 AES128
 * openssl pkcs8 -in client.key -topk8  -outform DER -out client.key.der -v2 aes-256-cbc -iter 1000000
 * openssl pkcs8 -in client.key -topk8 -out client.key.der  -outform Der -v2 aes-256-cbc -v2prf
hmacWithSHA512
 * Enable BouncyCastle: Introduce the bcpkix-jdk15on.jar package for projects that use JDBC. The
recommended version is 1.65 or later.
 */
```


NOTICE

When JDBC establishes a connection in SSL mode, a strong random number is obtained on the client. During the connection establishment, the error information shown in the figure may be displayed.



The random number generation on the client is too slow to meet product requirements. The entropy source is insufficient. As a result, the service fails to be started. This problem exists in some Linux environments.

Recommended solution: Start the haveged service on the client and increase the entropy value of the system entropy pool to improve the speed of reading random numbers. The startup command is as follows:

```
systemctl start haveged
```

5.3.6 Connecting to a Database (Using UDS)

The Unix domain socket is used for data exchange between different processes on the same host. You can add **junixsocket** to obtain the socket factory.

The **junixsocket-core-*XXX*.jar**, **junixsocket-common-*XXX*.jar**, and **junixsocket-native-common-*XXX*.jar** JAR packages need to be referenced. In addition, you need to add **socketFactory=org.newsclub.net.unix.AFUNIXSocketFactory$FactoryArg&socketFactoryArg=*[path-to-the-unix-socket]*** to the URL connection string.

Example:

```
// There will be security risks if the username and password used for authentication are directly written into code. It is recommended that the username and password be stored in the configuration file or environment variables (the password must be stored in ciphertext and decrypted when being used) to ensure security.
// In this example, the username and password are stored in environment variables. Before running this example, set environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in the local environment (set the environment variable names based on the actual situation).
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.util.Properties;

public class Test {
    public static void main(String[] args) {
        String driver = "com.huawei.gaussdb.jdbc.Driver";
        String userName = System.getenv("EXAMPLE_USERNAME_ENV");
```



```
        String password = System.getenv("EXAMPLE_PASSWORD_ENV");
        Connection conn;
        try {
            Class.forName(driver).newInstance();
            Properties properties = new Properties();
            properties.setProperty("user", userName);
            properties.setProperty("password", password);
            conn = DriverManager.getConnection("jdbc:gaussdb://$ip:$port/postgres?
socketFactory=org.newsclub" +
                    ".net.unix" +
                    ".AFUNIXSocketFactory$FactoryArg&socketFactoryArg=/data/tmp/.s.PGSQL.8000",
                properties);
            System.out.println("Connection Successful!");
            Statement statement = conn.createStatement();
            statement.executeQuery("select 1");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

NOTICE

- Set the **socketFactoryArg** parameter based on the actual path. The value must be the same as that of the GUC parameter **unix_socket_directory**.
- The connection host name must be set to **localhost**.

5.3.7 Running SQL Statements

Running a Common SQL Statement

To enable an application to operate data in the database by running SQL statements (statements that do not need to transfer parameters), perform the following operations:

Operations such as SELECT, UPDATE, INSERT, and DELETE can be performed on XML data.

Step 1 Create a statement object by calling the createStatement method in Connection.

```
// There will be security risks if the username and password used for authentication are directly written into code. It is recommended that the username and password be stored in the configuration file or environment variables (the password must be stored in ciphertext and decrypted when being used) to ensure security.
// In this example, the username and password are stored in environment variables. Before running this example, set environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in the local environment (set the environment variable names based on the actual situation).
String userName = System.getenv("EXAMPLE_USERNAME_ENV");
String password = System.getenv("EXAMPLE_PASSWORD_ENV");
Connection conn = DriverManager.getConnection("url",userName,password);
Statement stmt = conn.createStatement();
```

Step 2 Run the SQL statement by calling the executeUpdate method in Statement.

```
int rc = stmt.executeUpdate("CREATE TABLE customer_t1(c_customer_sk INTEGER, c_customer_name VARCHAR(32));");
```


NOTE

- If an execution request (not in a transaction block) received in the database contains multiple statements, the request is packed into a transaction. The VACUUM operation is not supported in a transaction block. If one of the statements fails, the entire request will be rolled back.
- Use semicolons (;) to separate statements. Stored procedures, functions, and anonymous blocks do not support multi-statement execution. When **preferQueryMode** is set to **simple**, the statement does not execute the parsing logic, and the semicolons (;) cannot be used to separate statements in this scenario.
- The slash (/) can be used as the terminator for creating a single stored procedure, function, anonymous block, or package body. When **preferQueryMode** is set to **simple**, the statement does not execute the parsing logic, and the slash (/) cannot be used as the terminator in this scenario.
- When **prepareThreshold** is set to **1**, each SQL statement executed by the statement is cached because cached statements are not evicted by default (default value of **preferQueryMode**). As a result, memory bloat may occur. In this case, set **preferQueryMode** to **extendedCacheEverything** to evict cached statements.

Step 3 Close the statement object.

```
stmt.close();
```

----End

Running a Prepared SQL Statement

Prepared statements are complied and optimized once but can be used in different scenarios by assigning multiple values. Using prepared statements improves execution efficiency. If you want to run a statement for several times, use a precompiled statement. Perform the following operations:

Step 1 Create a prepared statement object by calling the prepareStatement method in Connection.

```
PreparedStatement pstmt = con.prepareStatement("UPDATE customer_t1 SET c_customer_name = ? WHERE c_customer_sk = 1");
```

Step 2 Set parameters by calling the setShort method in PreparedStatement.

```
pstmt.setShort(1, (short)2);
```

CAUTION

After binding parameters are set in PrepareStatement, a B packet or U packet is constructed and sent to the server when the SQL statement is executed. However, the maximum length of a B packet or U packet cannot exceed 1023 MB. If the data bound at a time is too large, an exception may occur because the packet is too long. Therefore, when setting binding parameters in PrepareStatement, you need to evaluate and control the size of the bound data to avoid exceeding the upper limit of the packet.

Step 3 Run the prepared statement by calling the executeUpdate method in PreparedStatement.

```
int rowcount = pstmt.executeUpdate();
```

Step 4 Close the prepared statement object by calling the close method in PreparedStatement.


```
pstmt.close();
```

----End

Calling a Stored Procedure

To call an existing stored procedure through JDBC in GaussDB, perform the following operations:

Step 1 Create a call statement object by calling the prepareCall method in Connection.

```
// There will be security risks if the username and password used for authentication are directly written into code. It is recommended that the username and password be stored in the configuration file or environment variables (the password must be stored in ciphertext and decrypted when being used) to ensure security.
// In this example, the username and password are stored in environment variables. Before running this example, set environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in the local environment (set the environment variable names based on the actual situation).
String userName = System.getenv("EXAMPLE_USERNAME_ENV");
String password = System.getenv("EXAMPLE_PASSWORD_ENV");
Connection myConn = DriverManager.getConnection("url",userName,password);
CallableStatement cstmt = myConn.prepareCall("{? = CALL TESTPROC(?,?,?)}");
```

Step 2 Set parameters by calling the setInt method in CallableStatement.

```
cstmt.setInt(2, 50);
cstmt.setInt(1, 20);
cstmt.setInt(3, 90);
```

Step 3 Register an output parameter by calling the registerOutParameter method in CallableStatement.

```
cstmt.registerOutParameter(4, Types.INTEGER);  // Register an OUT parameter of the integer type.
```

Step 4 Call the stored procedure by calling the execute method in CallableStatement.

```
cstmt.execute();
```

Step 5 Obtain the output parameter by calling the getInt method in CallableStatement.

```
int out = cstmt.getInt(4);  // Obtain the OUT parameter.
```

Example:

```
// The following stored procedure (containing the OUT parameter) has been created:
create or replace procedure testproc
(
    psv_in1 in integer,
    psv_in2 in integer,
    psv_inout inout integer
)
as
begin
    psv_inout := psv_in1 + psv_in2 + psv_inout;
end;
/
```

Step 6 Close the call statement by calling the close method in CallableStatement.

```
cstmt.close();
```


NOTE

- Many database classes such as Connection, Statement, and ResultSet have a close() method. Close these classes after using their objects. Closing Connection will close all the related Statements, and closing a Statement will close its ResultSet.
- Some JDBC drivers support named parameters, which can be used to set parameters by name rather than sequence. If a parameter has the default value, you do not need to specify any parameter value but can use the default value directly. Even though the parameter sequence changes during a stored procedure, the application does not need to be modified. Currently, the GaussDB JDBC driver does not support this method.
- GaussDB does not support functions containing OUT parameters, or stored procedures and function parameters containing default values.
- When you bind parameters in **myConn.prepareCall("{? = CALL TESTPROC(?,?,?)}")** during a stored procedure calling, you can bind parameters and register the first parameter as the output parameter according to the placeholder sequence or the fourth parameter as the output parameter according to the parameter sequence in the stored procedure. The preceding example registers the fourth parameter.

NOTICE

- If JDBC is used to call a stored procedure whose returned value is a cursor, the returned cursor cannot be used.
- A stored procedure and an SQL statement must be run separately.
- Output parameters must be registered for parameters of the inout type in the stored procedure.

----End

Calling a Stored Procedure When Overloading Is Enabled in Oracle-Compatible Mode

Set **behavior_compat_options** to **'proc_outparam_override'**, and then perform the following steps to call the stored procedure based on JDBC:

Step 1 Create a call statement object by calling the prepareCall method in Connection.

```
// There will be security risks if the username and password used for authentication are directly written into code. It is recommended that the username and password be stored in the configuration file or environment variables (the password must be stored in ciphertext and decrypted when being used) to ensure security.
// In this example, the username and password are stored in environment variables. Before running this example, set environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in the local environment (set the environment variable names based on the actual situation).
String userName = System.getenv("EXAMPLE_USERNAME_ENV");
String password = System.getenv("EXAMPLE_PASSWORD_ENV");
Connection conn = DriverManager.getConnection("url",userName,password);
CallableStatement cs = conn.prepareCall("{ CALL TEST_PROC(?,?,?) }");
```

Step 2 Set parameters by calling the setInt method in CallableStatement.

```
PGobject pGobject = new PGobject();
pGobject.setType("public.compfoo"); // Set the composite type name. The format is "schema.typename".
pGobject.setValue("(1,demo)"); //: Bind the value of the composite type. The format is "(value1,value2)".
cs.setObject(1, pGobject);
```

Step 3 Register an output parameter by calling the registerOutParameter method in CallableStatement.

```
// Register an OUT parameter of the composite type. The format is "schema.typename".
cs.registerOutParameter(2, Types.STRUCT, "public.compfoo");
```



```
// Register an OUT parameter of the table type. The format is "schema.typename".
cs.registerOutParameter(3, Types.ARRAY, "public.compfoo_table");
```

Step 4 Call the stored procedure by calling the execute method in CallableStatement.

```
cs.execute();
```

Step 5 Obtain the output parameter by calling the getObject method in CallableStatement.

```
// The returned structure is of the user-defined type.
PGobject result = (PGobject)cs.getObject(2);  // Obtain the out parameter.
result.getValue(); // Obtain the string value of the composite type.
result.getArrayValue(); // Obtain the array values of the composite type and sort the values according to the sequence of columns of the composite type.
result.getStruct(); // Obtain the subtype names of the composite type and sort them according to the creation sequence.
result.getAttributes(); // Return the object constructed from data in each column of the user-defined type. For the array and table types, PgArray is returned. For the user-defined type, PGobject is encapsulated. For other types of data, a character string is returned.
// The returned result is of the table type.
PgArray pgArray = (PgArray) cs.getObject(3);
ResultSet rs = pgArray.getResultSet();
while (rs.next()) {
  rs.getObject(2);// Object constructed from data in each row of the table type.
}
```

NOTICE

If the table type of the output parameter is user-defined, for example, defined by running **create type compfooTable is table of compfoo**, the received return object is PgArray. In addition, the object obtained by **rs.getObject(2)** is also PgArray. In this case, the data of each column of the compfoo type cannot be obtained. To obtain the data, you must execute **getPGobject()** to obtain PgObject first.

Step 6 Close the call statement by calling the close method in CallableStatement.

```
cs.close();
```

NOTE

- After the Oracle-compatible mode is enabled, you must use the **{call proc_name(?,?,?)}** format to call a stored procedure and use the **{? = call func_name(?,?)}** format to call a function. The question mark (?) on the left of the equal mark is the placeholder for the return value of the function and is used to register the return value of the function.
- After **behavior_compat_options** is set to **'proc_outparam_override'**, the service needs to re-establish a connection. Otherwise, the stored procedures and functions cannot be correctly called.
- If a function or stored procedure contains a composite type, bind and register parameters in the schema.typename format.

----End

Example:

```
// Create a composite data type in the database.
CREATE TYPE compfoo AS (f1 int, f3 text);
// Create the table type in the database.
create type compfoo_table is table of compfoo;
// The following stored procedure (containing the OUT parameter) has been created:
create or replace procedure test_proc
(
    psv_in in compfoo,
    psv_out out compfoo,
```



```
    table_out out compfoo_table
)
as
begin
    psv_out := psv_in;
    table_out:=compfoo_table();
    table_out.extend(2);
    table_out(1):=psv_out;
    table_out(2):=psv_out;
end;
/
```

Batch Processing

When a prepared statement processes multiple pieces of similar data, the database creates only one execution plan. This improves compilation and optimization efficiency. Perform the following operations:

Step 1 Create a prepared statement object by calling the prepareStatement method in Connection.

```
// There will be security risks if the username and password used for authentication are directly written into code. It is recommended that the username and password be stored in the configuration file or environment variables (the password must be stored in ciphertext and decrypted when being used) to ensure security.
// In this example, the username and password are stored in environment variables. Before running this example, set environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in the local environment (set the environment variable names based on the actual situation).
String userName = System.getenv("EXAMPLE_USERNAME_ENV");
String password = System.getenv("EXAMPLE_PASSWORD_ENV");
Connection conn = DriverManager.getConnection("url",userName,password);
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO customer_t1 VALUES (?)");
```

Step 2 Call setShort to set parameters for each piece of data, and call addBatch to confirm that the setting is complete.

```
pstmt.setShort(1, (short)2);
pstmt.addBatch();
```

Step 3 Perform batch processing by calling the executeBatch method in PreparedStatement.

```
int[] rowcount = pstmt.executeBatch();
```

Step 4 Close the prepared statement object by calling the close method in PreparedStatement.

```
pstmt.close();
```

NOTE

Do not terminate a batch processing action when it is ongoing; otherwise, database performance will deteriorate. Therefore, disable automatic commit during batch processing. Manually commit several rows at a time. The statement for disabling automatic commit is **conn.setAutoCommit(false);**.

----End

5.3.8 Processing Data in a Result Set

Setting a Result Set Type

Different types of result sets apply to different application scenarios. Applications select proper types of result sets based on requirements. Before running an SQL statement, you must create a statement object. Some methods of creating

statement objects can set the type of a result set. **Table 5-5** lists result set parameters. The related Connection methods are as follows:

```
// Create a Statement object. This object will generate a ResultSet object with a specified type and concurrency.
createStatement(int resultSetType, int resultSetConcurrency);

// Create a PreparedStatement object. This object will generate a ResultSet object with a specified type and concurrency.
prepareStatement(String sql, int resultSetType, int resultSetConcurrency);

// Create a CallableStatement object. This object will generate a ResultSet object with a specified type and concurrency.
prepareCall(String sql, int resultSetType, int resultSetConcurrency);
```

Table 5-5 Result set types

Parameter	Description
resultSetType	Type of a result set. There are three types of result sets: • **ResultSet.TYPE_FORWARD_ONLY**: The ResultSet object can only be navigated forward. It is the default value. • **ResultSet.TYPE_SCROLL_SENSITIVE**: You can view the modified result by scrolling to the modified row. • **ResultSet.TYPE_SCROLL_INSENSITIVE**: The ResultSet object is insensitive to changes in the underlying data source. **NOTE** After a result set has obtained data from the database, the result set is insensitive to data changes made by other transactions, even if the result set type is **ResultSet.TYPE_SCROLL_SENSITIVE**. To obtain up-to-date data of the record pointed by the cursor from the database, call the refreshRow() method in a ResultSet object.
resultSetConcurrency	Concurrency type of a result set. There are two types of concurrency. • **ResultSet.CONCUR_READ_ONLY**: Data in a result set cannot be updated except that an updated statement has been created in the result set data. • **ResultSet.CONCUR_UPDATEABLE**: changeable result set. The concurrency type for a result set object can be updated if the result set is scrollable.

Positioning a Cursor in a Result Set

ResultSet objects include a cursor pointing to the current data row. The cursor is initially positioned before the first row. The next method moves the cursor to the next row from its current position. When a ResultSet object does not have a next row, a call to this method returns **false**. Therefore, this method is used in the while loop for result set iteration. However, the JDBC driver provides more cursor positioning methods for scrollable result sets, which allows positioning cursor in the specified row. **Table 5-6** describes these methods.

Table 5-6 Methods for positioning a cursor in a result set

Method	Description
next()	Moves cursor to the next row from its current position.
previous()	Moves cursor to the previous row from its current position.
beforeFirst()	Places cursor before the first row.
afterLast()	Places cursor after the last row.
first()	Places cursor to the first row.
last()	Places cursor to the last row.
absolute(int)	Places cursor to a specified row.
relative(int)	Moves the row specified by the forward parameter (that is, the value is **1**, which is equivalent to next()) or backward (that is, the value is **-1**, which is equivalent to previous()).

Obtaining the Cursor Position from a Result Set

This cursor positioning method can be used to change the cursor position for a scrollable result set. The JDBC driver provides a method to obtain the cursor position in a result set. **Table 5-7** describes these methods.

Table 5-7 Methods for obtaining a cursor position in a result set

Method	Description
isFirst()	Checks whether it is in the first row.
isLast()	Checks whether it is in the last row.
isBeforeFirst()	Checks whether it is before the first row.
isAfterLast()	Checks whether it is after the last row.
getRow()	Obtains its current row number.

Obtaining Data from a Result Set

ResultSet objects provide a variety of methods to obtain data from a result set. **Table 5-8** describes the common methods for obtaining data. If you want to know more about other methods, see JDK official documents.

Table 5-8 Common methods for obtaining data from a result set

Method	Description
int getInt(int columnIndex)	Retrieves the value of the column designated by a column index in the current row as an integer.
int getInt(String columnLabel)	Retrieves the value of the column designated by a column label in the current row as an integer.
String getString(int columnIndex)	Retrieves the value of the column designated by a column index in the current row as a string.
String getString(String columnLabel)	Retrieves the value of the column designated by a column label in the current row as a string.
Date getDate(int columnIndex)	Obtains date data by column index.
Date getDate(String columnLabel)	Retrieves the value of the column designated by a column name in the current row as a date.

5.3.9 Closing a Database Connection

After you complete required data operations in a database, close the database connection.

Call the close method to close the connection.

```
// There will be security risks if the username and password used for authentication are directly written into code. It is recommended that the username and password be stored in the configuration file or environment variables (the password must be stored in ciphertext and decrypted when being used) to ensure security.
// In this example, the username and password are stored in environment variables. Before running this example, set environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in the local environment (set the environment variable names based on the actual situation).
String userName = System.getenv("EXAMPLE_USERNAME_ENV");
String password = System.getenv("EXAMPLE_PASSWORD_ENV");
Connection conn = DriverManager.getConnection(sourceURL, userName, password);
conn.close();
```

5.3.10 Log Management

The GaussDB JDBC driver uses log records to help solve problems when the GaussDB JDBC driver is used in applications. GaussDB JDBC supports the following log management methods:

1. Use the SLF4J log framework for interconnecting with applications.
2. Use the JdkLogger log framework for interconnecting with applications.

SLF4J and JdkLogger are mainstream frameworks for Java application log management in the industry. For details about how to use these frameworks, see the official documents (SLF4J: **http://www.slf4j.org/manual.html**; JdkLogger:

https://docs.oracle.com/javase/8/docs/technotes/guides/logging/overview.html).

Method 1: Use the SLF4J log framework for interconnecting with applications.

When a connection is set up, **logger=Slf4JLogger** is configured in the URL.

The SLF4J may be implemented by using Log4j or Log4j2. When the Log4j is used to implement the SLF4J, the following JAR packages need to be added: **log4j-*.jar**, **slf4j-api-*.jar**, and **slf4j-log4*-*.jar** (* varies according to versions), and configuration file **log4j.properties**. In addition, the code for reading the configuration file must be added to the main program. If the Log4j2 is used to implement the SLF4J, you need to add the following JAR packages: **log4j-api-*.jar**, **log4j-core-*.jar**, **log4j-slf4j18-impl-*.jar**, and **slf4j-api-*-alpha1.jar** (* varies according to versions), and configuration file **log4j2.xml**.

This method supports log management and control. The SLF4J can implement powerful log management and control functions through related configurations in files. This method is recommended.

CAUTION

This method depends on the general SLF4J APIs, such as **org.slf4j.LoggerFactory.getLogger(String name)**, **org.slf4j.Logger.debug(String var1)**, **org.slf4j.Logger.info(String var1)**, **org.slf4j.Logger.warn(String warn)**, and **org.slf4j.Logger.warn(String warn)**. If these APIs are changed, logs cannot be recorded.

Example:

```
// Note: The location of the log4j.propertie or log4j2.xml file must be specified during calling.
// Specify the location of log4j.propertie.
//PropertyConfigurator.configure("log4j.properties");

// Specify log4j2.xml.
//ConfigurationSource source = new ConfigurationSource(new FileInputStream("log4j2.xml"));
//Configurator.initialize(null, source);

public static Connection GetConnection(String username, String passwd){

    String sourceURL = "jdbc:gaussdb://$ip:$port/database?logger=Slf4JLogger";
    Connection conn = null;

    try{
        // Create a connection.
        conn = DriverManager.getConnection(sourceURL,username,passwd);
        System.out.println("Connection succeed!");
    }catch (Exception e){
        e.printStackTrace();
        return null;
    }
    return conn;
}
```

The following is an example of the **log4j.properties** file:

```
log4j.logger.com.huawei.gaussdb.jdbc=ALL, log_gsjdbc

# Default file output configuration.
log4j.appender.log_gsjdbc=org.apache.log4j.RollingFileAppender
log4j.appender.log_gsjdbc.Append=true
```



```
log4j.appender.log_gsjdbc.File=gsjdbc.log
log4j.appender.log_gsjdbc.Threshold=TRACE
log4j.appender.log_gsjdbc.MaxFileSize=10MB
log4j.appender.log_gsjdbc.MaxBackupIndex=5
log4j.appender.log_gsjdbc.layout=org.apache.log4j.PatternLayout
log4j.appender.log_gsjdbc.layout.ConversionPattern=%d %p %t %c - %m%n
log4j.appender.log_gsjdbc.File.Encoding = UTF-8
```

The following is an example of the **log4j2.xml** file:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration status="OFF">
   <appenders>
      <Console name="Console" target="SYSTEM_OUT">
         <PatternLayout pattern="%d %p %t %c - %m%n"/>
      </Console>
      <File name="FileTest" fileName="test.log">
         <PatternLayout pattern="%d %p %t %c - %m%n"/>
      </File>
      <!-- JDBC driver log file output configuration. Log rewinding is supported. When the log size exceeds
10 MB, a new file is created. The new file is named in the format of yyyy-mm-dd-file ID. -->
      <RollingFile name="RollingFileJdbc" fileName="gsjdbc.log" filePattern="%d{yyyy-MM-dd}-%i.log">
         <PatternLayout pattern="%d %p %t %c - %m%n"/>
         <Policies>
            <SizeBasedTriggeringPolicy size="10 MB"/>
         </Policies>
      </RollingFile>
   </appenders>
   <loggers>
      <root level="all">
         <appender-ref ref="Console"/>
         <appender-ref ref="FileTest"/>
      </root>
      <!-- JDBC driver logs. The log level is all. All logs can be viewed and exported to the gsjdbc.log file. -->
      <logger name="com.huawei.gaussdb.jdbc" level="all" additivity="false">
         <appender-ref ref="RollingFileJdbc"/>
      </logger>
   </loggers>
</configuration>
```

Method 2: Use the JdkLogger log framework for interconnecting with applications.

The default Java logging framework stores its configurations in a file named **logging.properties**. Java installs the global configuration file in the folder in the Java installation directory. The **logging.properties** file can also be created and stored with a single project.

Configuration example of **logging.properties**:

```
# Specify the processing program as a file.
handlers= java.util.logging.FileHandler

# Specify the default global log level.
.level= ALL

# Specify the log output control standard.
java.util.logging.FileHandler.level=ALL
java.util.logging.FileHandler.pattern = gsjdbc.log
java.util.logging.FileHandler.limit = 500000
java.util.logging.FileHandler.count = 30
java.util.logging.FileHandler.formatter = java.util.logging.SimpleFormatter
java.util.logging.FileHandler.append=false
```

The following is a code example:

```
System.setProperty("java.util.logging.FileHandler.pattern","jdbc.log");
FileHandler fileHandler = new FileHandler(System.getProperty("java.util.logging.FileHandler.pattern"));
Formatter formatter = new SimpleFormatter();
fileHandler.setFormatter(formatter);
Logger logger = Logger.getLogger("com.huawei.gaussdb.jdbc");
```



```
logger.addHandler(fileHandler);
logger.setLevel(Level.ALL);
logger.setUseParentHandlers(false);
```

Link Trace Function

The GaussDB JDBC driver provides the application-to-database link trace function to associate discrete SQL statements on the database side with application requests. This function requires application developers to implement the com.huawei.gaussdb.jdbc.log.Tracer API class and specify the permission name of the API implementation class in the URL.

URL example:

```
String URL = "jdbc:gaussdb://$ip:$port/database?traceInterfaceClass=xxx.xxx.xxx.OpenGaussTraceImpl";
```

The com.huawei.gaussdb.jdbc.log.Tracer API class is defined as follows:

```
public interface Tracer {
// Retrieves the value of traceId.
String getTraceId();
}
```

The following is an example of the com.huawei.gaussdb.jdbc.log.Tracer API implementation class:

```
import com.huawei.gaussdb.jdbc.log.Tracer;

public class OpenGaussTraceImpl implements Tracer {
    private static MDC mdc = new MDC();

    private final String TRACE_ID_KEY = "traceId";

    public void set(String traceId) {
        mdc.put(TRACE_ID_KEY, traceId);
    }

    public void reset() {
        mdc.clear();
    }

    @Override
    public String getTraceId() {
        return mdc.get(TRACE_ID_KEY);
    }
}
```

The following is an example of context mapping which is used to store trace IDs generated for different requests:

```
import java.util.HashMap;

public class MDC {
    static final private ThreadLocal<HashMap<String, String>> threadLocal = new ThreadLocal<>();

    public void put(String key, String val) {
        if (key == null || val == null) {
            throw new IllegalArgumentException("key or val cannot be null");
        } else {
            if (threadLocal.get() == null) {
                threadLocal.set(new HashMap<>());
            }
            threadLocal.get().put(key, val);
        }
    }

    public String get(String key) {
        if (key == null) {
            throw new IllegalArgumentException("key cannot be null");
        } else if (threadLocal.get() == null) {
```



```
            return null;
        } else {
            return threadLocal.get().get(key);
        }
    }

    public void clear() {
        if (threadLocal.get() == null) {
            return;
        } else {
            threadLocal.get().clear();
        }
    }
}
```

Take the service **traceId** as an example. The prerequisite is that the table **test_trace_id (id int,name varchar(20))** is created.

```
String traceId = UUID.randomUUID().toString().replaceAll("-", "");
OpenGaussTraceImpl openGaussTrace = new OpenGaussTraceImpl();
openGaussTrace.set(traceId);
Connection con = DriverManager.getConnection(url, user, password);
pstm = con.prepareStatement("select * from test_trace_id where id = ?");
pstm.setInt(1, 1);
pstm.execute();
pstm = con.prepareStatement("insert into test_trace_id values(?,?)");
pstm.setInt(1, 2);
pstm.setString(2, "test");
pstm.execute();
openGaussTrace.reset();
```

NOTE

- When the link trace function is used, the link function at the application layer is guaranteed by services.
- The application must expose the API for obtaining **traceId** to the JDBC and configure the API implementation class to the JDBC connection string.
- SQL statements of the same request must use the same **traceId**.
- The value of **traceId** transferred by the application cannot exceed 32 bytes. Otherwise, the extra bytes will be truncated.

5.3.11 Examples: Common Operations

Example 1: Creating a Database Connection, Creating a Table, and Inserting Data

This example illustrates how to develop applications based on GaussDB JDBC APIs. Before executing the example, load the driver. For details about how to obtain and load the driver, see **JDBC Packages, Driver Classes, and Environment Classes**.

```
// DBTest.java
// This example illustrates the main processes of JDBC-based development, covering database connection creation, table creation, and data insertion.
// There will be security risks if the username and password used for authentication are directly written into code. It is recommended that the username and password be stored in the configuration file or environment variables (the password must be stored in ciphertext and decrypted when being used) to ensure security.
// In this example, the username and password are stored in environment variables. Before running this example, set environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in the local environment (set the environment variable names based on the actual situation).

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
```

```
import java.sql.Statement;
import java.sql.CallableStatement;
import java.sql.Types;

public class DBTest {

  // Create a database connection.
  public static Connection GetConnection(String username, String passwd) {
    String driver = "com.huawei.gaussdb.jdbc.Driver";
    String sourceURL = "jdbc:gaussdb://$ip:$port/database";
    Connection conn = null;
    try {
      // Load the database driver.
      Class.forName(driver).newInstance();
    } catch (Exception e) {
      e.printStackTrace();
      return null;
    }

    try {
      // Create a database connection.
      conn = DriverManager.getConnection(sourceURL, username, passwd);
      System.out.println("Connection succeed!");
    } catch (Exception e) {
      e.printStackTrace();
      return null;
    }

    return conn;
  };

  // Execute a common SQL statement to create table customer_t1.
  public static void CreateTable(Connection conn) {
    Statement stmt = null;
    try {
      stmt = conn.createStatement();

      // Execute common SQL statements.
      int rc = stmt
          .executeUpdate("CREATE TABLE customer_t1(c_customer_sk INTEGER, c_customer_name
VARCHAR(32));");

      stmt.close();
    } catch (SQLException e) {
      if (stmt != null) {
        try {
          stmt.close();
        } catch (SQLException e1) {
          e1.printStackTrace();
        }
      }
      e.printStackTrace();
    }
  }

  // Execute a prepared statement to insert data in batches.
  public static void BatchInsertData(Connection conn) {
    PreparedStatement pst = null;

    try {
      // Generate a prepared statement.
      pst = conn.prepareStatement("INSERT INTO customer_t1 VALUES (?,?)");
      for (int i = 0; i < 3; i++) {
        // Add parameters.
        pst.setInt(1, i);
        pst.setString(2, "data " + i);
        pst.addBatch();
      }
      // Perform batch processing.
```



```
    pst.executeBatch();
    pst.close();
  } catch (SQLException e) {
    if (pst != null) {
      try {
        pst.close();
      } catch (SQLException e1) {
      e1.printStackTrace();
      }
    }
    e.printStackTrace();
  }
}

// Run a prepared statement to update data.
public static void ExecPreparedSQL(Connection conn) {
  PreparedStatement pstmt = null;
  try {
    pstmt = conn
        .prepareStatement("UPDATE customer_t1 SET c_customer_name = ? WHERE c_customer_sk = 1");
    pstmt.setString(1, "new Data");
    int rowcount = pstmt.executeUpdate();
    pstmt.close();
  } catch (SQLException e) {
    if (pstmt != null) {
      try {
        pstmt.close();
      } catch (SQLException e1) {
        e1.printStackTrace();
      }
    }
    e.printStackTrace();
  }
}

// Create a stored procedure.
public static void CreateCallable(Connection conn) {
  Statement stmt = null;
  try {
    stmt = conn.createStatement();
    // Create a function to return the sum of the three input values.
    stmt.execute("create or replace procedure testproc \n" +
        "(\n" +
        "    psv_in1 in integer,\n" +
        "    psv_in2 in integer,\n" +
        "    psv_inout inout integer\n" +
        ")\n" +
        "as\n" +
        "begin\n" +
        "    psv_inout := psv_in1 + psv_in2 + psv_inout;\n" +
        "end;\n" +
        "/");
  } catch (SQLException e) {
    throw new RuntimeException(e);
  } finally {
    if (stmt != null) {
      try {
        stmt.close();
      } catch (SQLException e) {
        throw new RuntimeException(e);
      }
    }
  }
}

// Run a stored procedure.
public static void ExecCallableSQL(Connection conn) {
  CallableStatement cstmt = null;
  try {
```

```
    // The stored procedure TESTPROC must be created in advance.
    cstmt=conn.prepareCall("{? = CALL TESTPROC(?,?,?)}");
    cstmt.setInt(2, 50);
    cstmt.setInt(1, 20);
    cstmt.setInt(3, 90);
    cstmt.registerOutParameter(4, Types.INTEGER);  // Register an OUT parameter of the integer type.
    cstmt.execute();
    int out = cstmt.getInt(4);  // Obtain the OUT parameter.
    System.out.println("The CallableStatment TESTPROC returns:"+out);
    cstmt.close();
  } catch (SQLException e) {
    if (cstmt != null) {
      try {
        cstmt.close();
      } catch (SQLException e1) {
        e1.printStackTrace();
      }
    }
    e.printStackTrace();
  }
 }


 /**
  * Main process. Call static methods one by one.
  * @param args
 */
 public static void main(String[] args) {
   // Create a database connection.
   String userName = System.getenv("EXAMPLE_USERNAME_ENV");
   String password = System.getenv("EXAMPLE_PASSWORD_ENV");
   Connection conn = GetConnection(userName, password);

   // Create a table.
   CreateTable(conn);

   // Insert data in batches.
   BatchInsertData(conn);

   // Run a prepared statement to update data.
   ExecPreparedSQL(conn);

   // Create a stored procedure.
   CreateCallable(conn);

   // Run a stored procedure.
   ExecCallableSQL(conn);

   // Close the connection to the database.
   try {
     conn.close();
   } catch (SQLException e) {
     e.printStackTrace();
   }

 }

}
```

Example 2: High Client Memory Usage

In this example, **setFetchSize** adjusts the memory usage of the client by using the database cursor to obtain server data in batches. It may increase network interaction and damage some performance.

The cursor is valid within a transaction. Therefore, disable automatic commit and then manually commit the code.


```
// Disable automatic commit.
conn.setAutoCommit(false);

// Create a table.
Statement st = conn.createStatement();
st.execute("create table mytable (cal1 int);");

// Insert 200 rows of data into the table.
PreparedStatement pstm = conn.prepareStatement("insert into mytable values (?)");
for (int i = 0; i < 200; i++) {
    pstm.setInt(1, i + 1);
    pstm.addBatch();
}
pstm.executeBatch();
conn.commit();
pstm.close();

// Open the cursor and obtain 50 rows of data each time.
st.setFetchSize(50);
ResultSet rs = st.executeQuery("SELECT * FROM mytable");
while (rs.next()){
    System.out.println("a row was returned.");
}
conn.commit();
rs.close();

// Disable the server cursor.
st.setFetchSize(0);
rs = st.executeQuery("SELECT * FROM mytable");
while (rs.next()){
    System.out.println("many rows were returned.");
}
conn.commit();
rs.close();

// Close the statement.
st.close();
conn.close();
```

Enable automatic commit.

```
conn.setAutoCommit(true);
```

Example 3: Using Common Data Types

```
// Prerequisites
String createsql = "create table if not exists t_bit(col_bit bit)";
Statement stmt = conn.createStatement();
stmt.execute(createsql);
stmt.close();
// Example of using the bit type. Note that the value range of the bit type is [0,1].
Statement st = conn.createStatement();
String sqlstr = "create or replace function fun_1()\n" +
    "returns bit AS $$\n" +
    "select col_bit from t_bit limit 1;\n" +
    "$$\n" +
    "LANGUAGE SQL;";
st.execute(sqlstr);
CallableStatement c = conn.prepareCall("{ ? = call fun_1() }");
// Register the output type, which is a bit string.
c.registerOutParameter(1, Types.BIT);
c.execute();
// Use the Boolean type to obtain the result.
System.out.println(c.getBoolean(1));

// Example of using the float8 type
st.execute("create table if not exists t_float(col1 float8)");
PreparedStatement pstm = conn.prepareStatement("insert into t_float values(?)");
pstm.setDouble(1,123456.123);
```



```
pstm.execute();
pstm.close();

// Example of using the function whose return value is of the float8 type.
st.execute("create or replace function func_float() " +
    "return float8 " +
    "as declare " +
    "var1 float8; " +
    "begin " +
    " select col1 into var1 from t_float limit 1; " +
    " return var1; " +
    "end;");
CallableStatement cs = conn.prepareCall("{? = call func_float()}");
cs.registerOutParameter(1,Types.DOUBLE);
cs.execute();
System.out.println(cs.getDouble(1));
st.close();
```

Example 4: Using the Database Connection Monitoring Function

This example demonstrates how to use the connection monitoring function of the JDBC driver.

```
// gaussdbjdbc.jar is used as an example.
// There will be security risks if the username and password used for authentication are directly written into code. It is recommended that the username and password be stored in the configuration file or environment variables (the password must be stored in ciphertext and decrypted when being used) to ensure security.
// In this example, the username and password are stored in environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in the local environment (set the environment variable names based on the actual situation).
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class DBMonitorTest {
    // Create a database connection.
    public static void main(String[] args){
        String driver = "com.huawei.gaussdb.jdbc.Driver";
        String username = System.getenv("EXAMPLE_USERNAME_ENV");
        String passwd = System.getenv("EXAMPLE_PASSWORD_ENV");
        String sourceURL
            = "jdbc:gaussdb://$ip:$port/database?
dbMonitor=true&loggerLevel=debug&loggerFile=dbMonitor.log";
        try {
      // Load the database driver.
            Class.forName(driver).newInstance();
        } catch (Exception e) {
            e.printStackTrace();
        }

        Connection conn = null;
        Statement statement = null;
        try {
            // Create a database connection.
            conn = DriverManager.getConnection(sourceURL, username, passwd);

            // Create a table.
            statement = conn.createStatement();
            String createTableQuery = "CREATE TABLE IF NOT EXISTS mytable (id INT PRIMARY KEY, name
VARCHAR(50))";
            statement.executeUpdate(createTableQuery);

            // Insert data.
            String insertQuery = "INSERT INTO mytable (id, name) VALUES (1, 'John')";
            statement.executeUpdate(insertQuery);
```



```
            // Query data.
            String selectQuery = "SELECT * FROM mytable ";
            ResultSet resultSet = statement.executeQuery(selectQuery);
            while (resultSet.next()) {
                int id = resultSet.getInt("id");
                String name = resultSet.getString("name");
                System.out.println("id: " + id + ", name: " + name);
            }

            // Delete the table.
            String dropTableQuery = "DROP TABLE IF EXISTS mytable";
            statement.executeUpdate(dropTableQuery);
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (statement != null) {
                    statement.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        try {
            Thread.sleep(10000);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
}
```

You can view the following connection monitoring information in the **dbMonitor.log** file:

```
Nov 23, 2023 10:30:54 AM com.huawei.gaussdb.jdbc.qos.DataProcess saveQosResult
FINE: {
  "Destination host:port" : "localhost:8000",-- IP address and port number of the server.
  "Delay" : "1.00 ms",-- Network delay.
  "Jitter" : "0.04ms",-- Network jitter.
  "Loss" : "0%",-- Network packet loss rate.
  "DownloadSpeed" : "0.395Mbps",-- Downlink rate of the network.
  "UpLoadSpeed" : "0.498Mbps"-- Uplink rate of the network.
}

Nov 23, 2023 10:30:56 AM com.huawei.gaussdb.jdbc.CollectDBData saveCollectResult
FINE: {
  "openCount": "1",-- Number of times that the application enables database connections.
  "closeCount": "1",-- Number of times that the application disables database connections.
  "abortedCount": "0",-- Number of abnormal disconnections.
  "visitCount": "12",-- Number of access requests from applications to the database.
  "cpuUsage": "20.39%",-- CPU usage of the client.
  "memoryUsage": "98.32%"-- Memory usage of the client.
}
```

Example 5: Obtaining the Driver Version

```
Driver.getGSVersion();
```

5.3.12 Example: Retrying SQL Queries for Applications

If the primary database node is faulty and cannot be restored within 10s, GaussDB automatically switches the standby database node to the active state to ensure the normal running of the database. Jobs running during the failover will fail and

those started after the failover will not be affected. To prevent upper-layer services from being affected by the failover, refer to this example to construct an SQL retry mechanism at the service layer. Before executing the example, load the driver. For details about how to obtain and load the driver, see **JDBC Packages, Driver Classes, and Environment Classes**.

```
// There will be security risks if the username and password used for authentication are directly written into code. It is recommended that the username and password be stored in the configuration file or environment variables (the password must be stored in ciphertext and decrypted when being used) to ensure security.
// In this example, the username and password are stored in environment variables. Before running this example, set environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in the local environment (set the environment variable names based on the actual situation).
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

class ExitHandler extends Thread {
    private Statement cancel_stmt = null;

    public ExitHandler(Statement stmt) {
        super("Exit Handler");
        this.cancel_stmt = stmt;
    }
    public void run() {
        System.out.println("exit handle");
        try {
            this.cancel_stmt.cancel();
        } catch (SQLException e) {
            System.out.println("cancel query failed.");
            e.printStackTrace();
        }
    }
}

public class SQLRetry {
   // Create a database connection.
   public static Connection GetConnection(String username, String passwd) {
     String driver = "com.huawei.gaussdb.jdbc.Driver";
     String sourceURL = "jdbc:gaussdb://$ip:$port/database";
     Connection conn = null;
     try {
       // Load the database driver.
       Class.forName(driver).newInstance();
     } catch (Exception e) {
       e.printStackTrace();
       return null;
     }

     try {
       // Create a database connection.
       conn = DriverManager.getConnection(sourceURL, username, passwd);
       System.out.println("Connection succeed!");
     } catch (Exception e) {
       e.printStackTrace();
       return null;
     }

     return conn;
}

   // Run a common SQL statement to create the jdbc_test1 table.
   public static void CreateTable(Connection conn) {
     Statement stmt = null;
     try {
```



```
    stmt = conn.createStatement();


    Runtime.getRuntime().addShutdownHook(new ExitHandler(stmt));

    // Execute common SQL statements.
    int rc2 = stmt
       .executeUpdate("DROP TABLE if exists jdbc_test1;");

    int rc1 = stmt
       .executeUpdate("CREATE TABLE jdbc_test1(col1 INTEGER, col2 VARCHAR(10));");

    stmt.close();
  } catch (SQLException e) {
    if (stmt != null) {
      try {
        stmt.close();
      } catch (SQLException e1) {
        e1.printStackTrace();
      }
    }
    e.printStackTrace();
  }
}

// Execute a prepared statement to insert data in batches.
public static void BatchInsertData(Connection conn) {
  PreparedStatement pst = null;

  try {
    // Generate a prepared statement.
    pst = conn.prepareStatement("INSERT INTO jdbc_test1 VALUES (?,?)");
    for (int i = 0; i < 100; i++) {
     // Add parameters.
      pst.setInt(1, i);
      pst.setString(2, "data " + i);
      pst.addBatch();
    }
    // Perform batch processing.
    pst.executeBatch();
    pst.close();
  } catch (SQLException e) {
    if (pst != null) {
      try {
        pst.close();
      } catch (SQLException e1) {
      e1.printStackTrace();
      }
    }
    e.printStackTrace();
  }
}

// Run a prepared statement to update data.
private static boolean QueryRedo(Connection conn){
  PreparedStatement pstmt = null;
  boolean retValue = false;
  try {
    pstmt = conn
        .prepareStatement("SELECT col1 FROM jdbc_test1 WHERE col2 = ?");

        pstmt.setString(1, "data 10");
        ResultSet rs = pstmt.executeQuery();

        while (rs.next()) {
            System.out.println("col1 = " + rs.getString("col1"));
        }
        rs.close();
```



```
      pstmt.close();
       retValue = true;
     } catch (SQLException e) {
      System.out.println("catch..... retValue " + retValue);
      if (pstmt != null) {
        try {
         pstmt.close();
       } catch (SQLException e1) {
         e1.printStackTrace();
        }
      }
      e.printStackTrace();
    }

     System.out.println("finesh.....");
    return retValue;
  }

  // Configure the number of retry attempts for the retry of a query statement upon a failure.
  public static void ExecPreparedSQL(Connection conn) throws InterruptedException {
        int maxRetryTime = 50;
        int time = 0;
        String result = null;
        do {
            time++;
            try {
System.out.println("time:" + time);
boolean ret = QueryRedo(conn);
if(ret == false){
 System.out.println("retry, time:" + time);
 Thread.sleep(10000);
 QueryRedo(conn);
}
            } catch (Exception e) {
                e.printStackTrace();
            }
        } while (null == result && time < maxRetryTime);

  }

  /**
   *Main process. Call static methods one by one.
   * @param args
   * @throws InterruptedException
   */
  public static void main(String[] args) throws InterruptedException {
    // Create a database connection.
    String userName = System.getenv("EXAMPLE_USERNAME_ENV");
    String password = System.getenv("EXAMPLE_PASSWORD_ENV");
    Connection conn = GetConnection(userName, password);

    // Create a table.
    CreateTable(conn);

    // Insert data in batches.
    BatchInsertData(conn);

    // Run a prepared statement to update data.
    ExecPreparedSQL(conn);

    // Close the connection to the database.
    try {
      conn.close();
    } catch (SQLException e) {
      e.printStackTrace();
    }

  }
```

```
}
```

5.3.13 Example: Importing and Exporting Data Through Local Files

When Java is used for secondary development based on GaussDB, you can use the CopyManager API to export data from the database to a local file or import a local file to the database by streaming. The file can be in CSV or TEXT format.

The example is as follows. Load the GaussDB JDBC driver before executing it. For details about how to obtain and load the driver, see **JDBC Packages, Driver Classes, and Environment Classes**.

Prerequisites: Tables **migration_table** and **migration_table_1** have been created in the database, and data has been inserted into the **migration_table** table.

```
// There will be security risks if the username and password used for authentication are directly written into code. It is recommended that the username and password be stored in the configuration file or environment variables (the password must be stored in ciphertext and decrypted when being used) to ensure security.
// In this example, the username and password are stored in environment variables. Before running this example, set environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in the local environment (set the environment variable names based on the actual situation).
import java.sql.Connection;
import java.sql.DriverManager;
import java.io.IOException;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.sql.SQLException;
import com.huawei.gaussdb.jdbc.copy.CopyManager;
import com.huawei.gaussdb.jdbc.core.BaseConnection;

public class Copy{

    public static void main(String[] args)
    {
     String urls = new String("jdbc:gaussdb://$ip:$port/database"); // Database URL
     String username = System.getenv("EXAMPLE_USERNAME_ENV");           // Username
     String password = System.getenv("EXAMPLE_PASSWORD_ENV");             // Password
     String tablename = new String("migration_table"); // Table information
     String tablename1 = new String("migration_table_1"); // Table information
     String driver = "com.huawei.gaussdb.jdbc.Driver";
     Connection conn = null;

     try {
           Class.forName(driver);
           conn = DriverManager.getConnection(urls, username, password);
         } catch (ClassNotFoundException e) {
             e.printStackTrace(System.out);
         } catch (SQLException e) {
             e.printStackTrace(System.out);
         }

     // Export the query result of SELECT * FROM migration_table to the local file d:/data.txt.
     try {
    copyToFile(conn, "d:/data.txt", "(SELECT * FROM migration_table)");
  } catch (SQLException e) {

 e.printStackTrace();
  } catch (IOException e) {

 e.printStackTrace();
  }
     // Import data from the d:/data.txt file to the migration_table_1 table.
     try {
```

```
    copyFromFile(conn, "d:/data.txt", tablename1);
  } catch (SQLException e) {
       e.printStackTrace();
} catch (IOException e) {

 e.printStackTrace();
}

    // Export the data from the migration_table_1 table to the d:/data1.txt file.
    try {
    copyToFile(conn, "d:/data1.txt", tablename1);
  } catch (SQLException e) {

 e.printStackTrace();
  } catch (IOException e) {


 e.printStackTrace();
}
   }
 // Use copyIn to import data from a file to the database.
 public static void copyFromFile(Connection connection, String filePath, String tableName)
       throws SQLException, IOException {

   FileInputStream fileInputStream = null;

   try {
      CopyManager copyManager = new CopyManager((BaseConnection)connection);
      fileInputStream = new FileInputStream(filePath);
      copyManager.copyIn("COPY " + tableName + " FROM STDIN", fileInputStream);
   } finally {
      if (fileInputStream != null) {
         try {
            fileInputStream.close();
         } catch (IOException e) {
            e.printStackTrace();
         }
      }
   }
}

 // Use copyOut to export data from the database to a file.
 public static void copyToFile(Connection connection, String filePath, String tableOrQuery)
       throws SQLException, IOException {

    FileOutputStream fileOutputStream = null;

    try {
       CopyManager copyManager = new CopyManager((BaseConnection)connection);
       fileOutputStream = new FileOutputStream(filePath);
       copyManager.copyOut("COPY " + tableOrQuery + " TO STDOUT", fileOutputStream);
    } finally {
       if (fileOutputStream != null) {
          try {
             fileOutputStream.close();
          } catch (IOException e) {
             e.printStackTrace();
          }
       }
    }
 }
}
```

5.3.14 Example: Migrating Data from MY

This example shows how to use CopyManager to migrate data from MY to GaussDB. Before executing the example, load the driver. For details about how to

obtain and load the driver, see **JDBC Packages, Driver Classes, and Environment Classes**.

```
// There will be security risks if the username and password used for authentication are directly written into code. It is recommended that the username and password be stored in the configuration file or environment variables (the password must be stored in ciphertext and decrypted when being used) to ensure security.
// In this example, the username and password are stored in environment variables. Before running this example, set environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in the local environment (set the environment variable names based on the actual situation).
import java.io.StringReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import com.huawei.gaussdb.jdbc.copy.CopyManager;
import com.huawei.gaussdb.jdbc.core.BaseConnection;

public class Migration{

    public static void main(String[] args) {
        String url = new String("jdbc:gaussdb://$ip:$port/database"); // Database URL
        String user = System.getenv("EXAMPLE_USERNAME_ENV");            // GaussDB username
        String pass = System.getenv("EXAMPLE_PASSWORD_ENV");            // GaussDB password
        String tablename = new String("migration_table"); // Table information
        String delimiter = new String("|");              // Delimiter
        String encoding = new String("UTF8");            // Character set
        String driver = "com.huawei.gaussdb.jdbc.Driver";
        StringBuffer buffer = new StringBuffer();       // Buffer to store formatted data

        try {
            // Obtain the query result set of the source database.
            ResultSet rs = getDataSet();

            // Traverse the result set and obtain records row by row.
            // The values of columns in each record are separated by the specified delimiter and end with a linefeed, forming strings.
            // Add the strings to the buffer.
            while (rs.next()) {
                buffer.append(rs.getString(1) + delimiter
                        + rs.getString(2) + delimiter
                        + rs.getString(3) + delimiter
                        + rs.getString(4)
                        + "\n");
            }
            rs.close();

            try {
                // Connect to the target database.
                Class.forName(driver);
                Connection conn = DriverManager.getConnection(url, user, pass);
                BaseConnection baseConn = (BaseConnection) conn;
                baseConn.setAutoCommit(false);

                // Initialize the table.
                String sql = "Copy " + tablename + " from STDIN DELIMITER " + "'" + delimiter + "'" + " ENCODING " + "'" + encoding + "'";

                // Commit data in the buffer.
                CopyManager cp = new CopyManager(baseConn);
                StringReader reader = new StringReader(buffer.toString());
                cp.copyIn(sql, reader);
                baseConn.commit();
                reader.close();
                baseConn.close();
            } catch (ClassNotFoundException e) {
                e.printStackTrace(System.out);
```



```
            } catch (SQLException e) {
                e.printStackTrace(System.out);
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    //********************************
    // Return the query result set from the source database.
    //********************************
    private static ResultSet getDataSet() {
        ResultSet rs = null;
        try {
            Class.forName("com.MySQL.jdbc.Driver").newInstance();
            String userName = System.getenv("EXAMPLE_USERNAME_ENV");
            String password = System.getenv("EXAMPLE_PASSWORD_ENV");
            Connection conn = DriverManager.getConnection("jdbc:MySQL://$ip:$port/database?
useSSL=false&allowPublicKeyRetrieval=true", userName, password);
            Statement stmt = conn.createStatement();
            rs = stmt.executeQuery("select * from migration_table");
        } catch (SQLException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return rs;
    }
}
```

5.3.15 Example: Logic Replication Code

The example demonstrates how to use the logical replication function through the JDBC APIs. Before executing the example, load the driver. For details about how to obtain and load the driver, see **JDBC Packages, Driver Classes, and Environment Classes**.

For details about the configuration options of logical replication, see section "Logical Replication > Logical Decoding" in *Feature Guide*. In addition, the following configuration items are added for streaming decoding tools such as JDBC:

1. Decoding thread concurrency

   Set **parallel-decode-num** to specify the number of decoder threads for parallel decoding. The value is an integer ranging from 1 to 20. The value **1** indicates that decoding is performed based on the original serial logic. Other values indicate that parallel decoding is enabled. The default value is **1**. When this parameter is set to **1**, do not configure the following options: **decode-style**, **sending-batch**, and **parallel-queue-size**.

2. Decoding format

   Configure **decode-style** to specify the decoding format. The value can be **'j'**, **'t'** or **'b'** of the char type, indicating the JSON, text, or binary format, respectively. The default value is **'b'**, indicating binary decoding. This option is set only when parallel decoding is allowed and binary decoding is supported only in the parallel decoding scenario. For the JSON and text formats corresponding to the binary format, in the decoding result sent in batches, the uint32 consisting of the first four bytes of each decoding statement indicates the total number of bytes of the statement (the four bytes occupied by the uint32 are excluded, and **0** indicates that the decoding of this batch ends).

The 8-byte uint64 indicates the corresponding LSN (**begin** corresponds to **first_lsn**, **commit** corresponds to **end_lsn**, and other values correspond to the LSN of the statement).

NOTE

The binary encoding rules are as follows:

1. The first four bytes represent the total number of bytes of the decoding result of statements following the statement-level delimiter letter P (excluded) or the batch end character F (excluded). If the value is **0**, the decoding of this batch ends.
2. The next eight bytes (uint64) indicate the corresponding LSN (**begin** corresponds to **first_lsn**, **commit** corresponds to **end_lsn**, and other values correspond to the LSN of the statement).
3. The next one-byte letter can be **B**, **C**, **I**, **U**, or **D**, representing BEGIN, COMMIT, INSERT, UPDATE, or DELETE.
4. When the one-byte letter in step 3 is B:
    1. The next eight bytes (uint64) indicate the CSN.
    2. The next eight bytes (uint64) indicate **first_lsn**.
    3. (Optional) If the next 1-byte letter is **T**, the following four bytes (uint32) indicate the timestamp length for committing the transaction. The following characters with the same length are the timestamp character string.
    4. (Optional) If the next one-byte letter is **N**, the following four bytes (uint32) indicate the length of the transaction username. The following characters with the same length are the transaction username.
    5. Because there may still be a decoding statement subsequently, a 1-byte letter **P** or **F** is used as a separator between statements. **P** indicates that there are still decoded statements in this batch, and **F** indicates that decoding in this batch is complete.
5. When the one-byte letter in step 3 is C:
    1. (Optional) If the next 1-byte letter is **X**, the following eight bytes (uint64) indicate XID.
    2. (Optional) If the next 1-byte letter is **T**, the following four bytes (uint32) indicate the timestamp length. The following characters with the same length are the timestamp character string.
    3. When logs are sent in batches, decoding results of other transactions may still exist after a COMMIT log is decoded. If the next 1-byte letter is **P**, the batch still needs to be decoded. If the letter is **F**, the batch decoding ends.
6. When the one-byte letter in step 3 is I, U, or D:
    1. The following two bytes (uint16) indicate the length of the schema name.
    2. The schema name is read based on the preceding length.
    3. The following two bytes (uint16) indicate the length of the table name.
    4. The table name is read based on the preceding length.
    5. (Optional) If the next 1-byte letter is **N**, it indicates a new tuple. If the letter is **O**, it indicates an old tuple. In this case, the new tuple is sent first.
        1. The following two bytes (uint16) indicate the number of columns to be decoded for the tuple, which is recorded as **attrnum**.
        2. The following procedure is repeated for *attrnum* times:
            1. The next two bytes (uint16) indicate the length of the column name.
            2. The column name is read based on the preceding length.
            3. The following four bytes (uint32) indicate the OID of the current column type.
            4. The next four bytes (uint32) indicate the length of the value (stored in the character string format) in the current column. If the value is **0xFFFFFFFF**, it indicates null. If the value is **0**, it indicates a character string whose length is 0.
            5. The column value is read based on the preceding length.

6. Because there may still be a decoding statement after, if the next one-byte letter is **P**, it indicates that the batch still needs to be decoded, and if the next one-byte letter is **F**, it indicates that decoding of the batch ends.

3. Decoding only on the standby node

   Configure the **standby-connection** option to specify whether to perform decoding only on the standby node. The value is of the Boolean type (**0** or **1**). The value **true** (or **1**) indicates that only the standby node can be connected for decoding. When the primary node is connected for decoding, an error is reported and the system exits. The value **false** (or **0**) indicates that there is no restriction. The default value is **false** (or **0**).

4. Batch sending

   Configure **sending-batch** to specify whether to send results in batches. The value is an integer ranging from 0 to 1. The value **0** indicates that decoding results are sent one by one. The value **1** indicates that decoding results are sent in batches when the accumulated size of decoding results reaches 1 MB. The default value is **0**. This parameter can be set only during parallel decoding. In the scenario where batch sending is enabled, if the decoding format is 'j' or 't', before each original decoding statement, a uint32 type is added indicating the length of the decoding result (excluding the current uint32 type), and a uint64 type is added, indicating the LSN corresponding to the current decoding result.

5. Length of the parallel decoding queue

   Configure **parallel-queue-size** to specify the length of the queue for interaction among parallel logical decoding threads. The value ranges from 2 to 1024 and must be a power of 2. The default value is **128**. The queue length is positively correlated with the memory usage during decoding.

6. Memory threshold for logical decoding

   The **max-txn-in-memory** configuration item specifies the memory threshold for caching the intermediate decoding result of a single transaction, in MB. The value range is [0,100] for serial decoding. The default value is **0**, indicating that the memory usage is not controlled. For parallel decoding, the value range is [0,*max_process_memory* x 25%]. The default value is *max_process_memory*/**4/1024**, where **1024** indicates the conversion from KB to MB. The value **0** indicates that this memory control is disabled. The **max-reorderbuffer-in-memory** configuration item specifies the memory threshold for caching intermediate decoding results of all transactions, in GB. The value range is [0,100] for serial decoding. The default value is **0**, indicating that the memory usage is not controlled. For parallel decoding, the value range is [0, *max_process_memory* x 50%]. The default value is *max_process_memory*/**2/1048576**, where **1048576** indicates the conversion from KB to GB. The value **0** indicates that this memory control is disabled. When the memory usage exceeds the threshold, intermediate decoding results are written into a temporary file during decoding, affecting the logical decoding performance.

7. Logical decoding sending timeout threshold

   The **sender-timeout** configuration item specifies the heartbeat timeout threshold between the kernel and client. If no message is received from the client within the period, the logical decoding stops and disconnects from the client. The unit is ms, and the value range is [0,2147483647]. The default value depends on the value of the GUC parameter **logical_sender_timeout**.

8. User blacklist options for logical decoding

   Use the user blacklist for logical decoding. The transaction operations of blacklisted users are filtered from the logic decoding result. The options are as follows:

   a. **exclude-userids**: specifies the OIDs of blacklisted users. Multiple OIDs are separated by commas (,). The system does not check whether the user OIDs exist.

   b. **exclude-users**: specifies blacklisted usernames. Multiple usernames are separated by commas (,). **dynamic-resolution** specifies whether to dynamically parse and identify usernames. If the decoding is interrupted because the user does not exist and the corresponding blacklisted user does not exist at the time when logs are generated, you can set **dynamic-resolution** to **true** or delete the username from the blacklist to start decoding and continue to obtain logical logs.

   c. **dynamic-resolution**: specifies whether to dynamically parse blacklisted usernames. The default value is **true**. If the parameter is set to **false**, an error is reported and the logic decoding exits when the decoding detects that the user does not exist in blacklist **exclude-users**. If the parameter is set to **true**, decoding continues when it detects that the user does not exist in blacklist **exclude-users**.

9. Output options for transaction logic logs

   a. **include-xids**: specifies whether the BEGIN logical log of a transaction outputs the transaction ID. The default value is **true**.

   b. **include-timestamp**: specifies whether the BEGIN logical log of a transaction outputs the time when the transaction is committed. The default value is **false**.

   c. **include-user**: specifies whether the BEGIN logical log of a transaction outputs the username of the transaction. The default value is **false**. The username of a transaction refers to the authorized user, that is, the login user who executes the session corresponding to the transaction. The username does not change during the execution of the transaction.

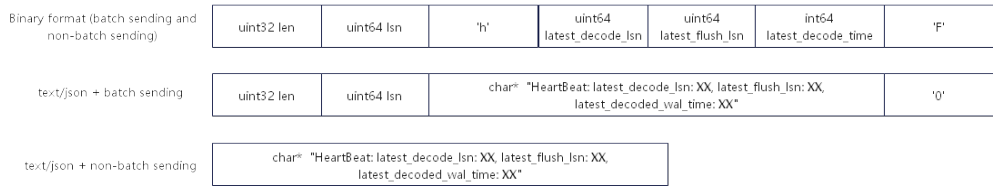
10. By default, **socketTimeout** of the logical decoding connection is set to **10s**. When the primary node is overloaded during decoding on the standby node, the connection may be closed due to timeout. You can set **withStatusInterval(10000,TimeUnit.MILLISECONDS)** to adjust the timeout interval.

11. Heartbeat log output option

    enable-heartbeat: specifies whether to generate heartbeat logs. The default value is **false**.

NOTE

If the heartbeat log output option is enabled, heartbeat logs will be generated. The heartbeat logs can be in the binary or TEXT/JSON format. For a binary heartbeat log message, it starts with a character 'h' and then the heartbeat log content: an 8-byte uint64 LSN indicating the end position of WAL reading when the heartbeat logical log is sent, an 8-byte uint64 LSN indicating the location of the WAL that has been flushed to disks when the heartbeat logical log is sent, and an 8-byte int64 timestamp (starting from January 1, 1970) indicating the timestamp when the latest decoded transaction log or checkpoint log is generated; then, it ends with character 'F'. TEXT/JSON heartbeat log messages that are sent in batches end with '0'. There is no such terminator for each TEXT/JSON heartbeat log message. The message content is transmitted in big-endian mode. The following figure shows the format.



12. Specifies the DDL statement reverse parsing process and output format for logical decoding.
    a. **enable-ddl-decoding**: The default value is **false**, indicating that logical decoding of DDL statements is disabled. The value **true** indicates that logical decoding of DDL statements is enabled.
    b. **enable-ddl-json-format**: The default value is **false**, indicating that the DDL statement reverse parsing result is output in TEXT format. The value **true** indicates that the DDL statement reverse parsing result is output in JSON format.

The decoding performance (Xlog consumption) is greater than or equal to 100 Mbps in the following standard parallel decoding scenario: 16-core CPU, 128 GB memory, network bandwidth > 200 Mbps, 10 to 100 columns in a table, 0.1 KB to 1 KB data in a single row, INSERT as main DML operations, less than 4096 statements in a single transaction, **parallel-decode-num** set to **8**, decoding format as **'t'**, and batch sending function enabled. To ensure that the decoding performance meets the requirements and minimize the impact on services, you are advised to set up only one parallel decoding connection on a standby node to ensure that the CPU, memory, and bandwidth resources are sufficient.

CAUTION

The logical replication class PGReplicationStream is a non-thread-safe class. Concurrent calls may cause data exceptions.

```
// There will be security risks if the username and password used for authentication are directly written into code. It is recommended that the username and password be stored in the configuration file or environment variables (the password must be stored in ciphertext and decrypted when being used) to ensure security.
// In this example, the username and password are stored in environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in the local environment (set the environment variable names based on the actual situation).
// Logical replication function example: file name, LogicalReplicationDemo.java
// Prerequisite: Add the IP address of the JDBC user machine to the database whitelist. Add the following content to gs_hba.conf:
// Assume that the IP address of the JDBC user machine is 10.10.10.10.
```

```
//host    all             all             10.10.10.10/32        sha256
//host    replication     all             10.10.10.10/32        sha256

import com.huawei.gaussdb.jdbc.PGProperty;
import com.huawei.gaussdb.jdbc.jdbc.PgConnection;
import com.huawei.gaussdb.jdbc.replication.LogSequenceNumber;
import com.huawei.gaussdb.jdbc.replication.PGReplicationStream;

import java.nio.ByteBuffer;
import java.sql.DriverManager;
import java.util.Properties;
import java.util.concurrent.TimeUnit;

public class LogicalReplicationDemo {
    private static PgConnection conn = null;
    public static void main(String[] args) {
        String driver = "com.huawei.gaussdb.jdbc.Driver";
 // Configure the IP address and haPort number of the database. By default, the port number is the port number of the connected DN plus 1.
        String sourceURL = "jdbc:gaussdb://$ip:$port/database";
 // The default name of the logical replication slot is replication_slot.
 // Test mode: Create a logical replication slot.
        int TEST_MODE_CREATE_SLOT = 1;
 // Test mode: Enable logical replication (the prerequisite is that the logical replication slot already exists).
        int TEST_MODE_START_REPL = 2;
 // Test mode: Delete a logical replication slot.
        int TEST_MODE_DROP_SLOT = 3;
        // Enable different test modes.
        int testMode = TEST_MODE_START_REPL;

        try {
            Class.forName(driver);
        } catch (Exception e) {
            e.printStackTrace();
            return;
        }

        try {
            Properties properties = new Properties();
            PGProperty.USER.set(properties, System.getenv("EXAMPLE_USERNAME_ENV"));
            PGProperty.PASSWORD.set(properties, System.getenv("EXAMPLE_PASSWORD_ENV"));
    // For logical replication, the following three attributes are mandatory:
            PGProperty.ASSUME_MIN_SERVER_VERSION.set(properties, "9.4");
            PGProperty.REPLICATION.set(properties, "database");
            PGProperty.PREFER_QUERY_MODE.set(properties, "simple");
            conn = (PgConnection) DriverManager.getConnection(sourceURL, properties);
            System.out.println("connection success!");

            if(testMode == TEST_MODE_CREATE_SLOT){
                conn.getReplicationAPI()
                        .createReplicationSlot()
                        .logical()
                        .withSlotName("replication_slot") // If the character string contains uppercase letters, the uppercase letters are automatically converted to lowercase letters.
                        .withOutputPlugin("mppdb_decoding")
                        .make();
            }else if(testMode == TEST_MODE_START_REPL) {
                // Create a replication slot before enabling this mode.
                LogSequenceNumber waitLSN = LogSequenceNumber.valueOf("6F/E3C53568");
                PGReplicationStream stream = conn
                        .getReplicationAPI()
                        .replicationStream()
                        .logical()
                        .withSlotName("replication_slot")
                        .withSlotOption("include-xids", false)
                        .withSlotOption("skip-empty-xacts", true)
                        .withStartPosition(waitLSN)
                        .withSlotOption("parallel-decode-num", 10) // Decoding thread concurrency
                        .withSlotOption("white-table-list", "public.t1,public.t2") // Whitelist
```



```
                    .withSlotOption("standby-connection", true) // Forcible standby decoding
                    .withSlotOption("decode-style", "t") // Decoding format
                    .withSlotOption("sending-batch", 0) // Sending decoding results in batches
                    .withSlotOption("max-txn-in-memory", 100) // The memory threshold for flushing a single decoding transaction to disks is 100 MB.
                    .withSlotOption("max-reorderbuffer-in-memory", 2) // The total memory threshold for flushing decoding transactions that are being handled to disks is 2 GB.
                    .withSlotOption("exclude-users", "userA") // The logical log of the transaction executed by user A is not returned.
                    .withSlotOption("include-user", true) // The BEGIN logical log of the transaction contains the username.
                    .withSlotOption("enable-heartbeat", true) // Enable the heartbeat log output option.
                    .start();
            while (true) {
                ByteBuffer byteBuffer = stream.readPending();

                if (byteBuffer == null) {
                    TimeUnit.MILLISECONDS.sleep(10L);
                    continue;
                }

                int offset = byteBuffer.arrayOffset();
                byte[] source = byteBuffer.array();
                int length = source.length - offset;
                System.out.println(new String(source, offset, length));

                // If the LSN needs to be flushed, call the following APIs based on the service requirements:
                //LogSequenceNumber lastRecv = stream.getLastReceiveLSN();
                //stream.setFlushedLSN(lastRecv);
                //stream.forceUpdateStatus();

            }
        }else if(testMode == TEST_MODE_DROP_SLOT){
            conn.getReplicationAPI()
                    .dropReplicationSlot("replication_slot");
        }
    } catch (Exception e) {
        e.printStackTrace();
        return;
    }
  }
}
// Note: The preceding code cannot directly read the logical logs in binary format. You need to read the logical logs according to the binary encoding rules.
```

The following is an example of the decoding result in text format (that is, 't' format):

```
BEGIN CSN: 2014 first_lsn: 0/2816A28
table public t1 INSERT: a[integer]:1 b[integer]:2 c[text]:'hello'
COMMIT XID: 15504
BEGIN CSN: 2015 first_lsn: 0/2816C20
table public t1 UPDATE: old-key: a[integer]:1 new-tuple: a[integer]:1 b[integer]:5 c[text]:'hello'
COMMIT XID: 15505
BEGIN CSN: 2016 first_lsn: 0/2816D60
table public t1 DELETE: a[integer]:1
COMMIT XID: 15506
```

The following is an example of the decoding result in JSON format (that is, 'j' format):

```
BEGIN CSN: 2014 first_lsn: 0/2816A28
{"table_name":"public.t1","op_type":"INSERT","columns_name":["a","b","c"],"columns_type":["integer","integer","text"],"columns_val":["1","1","'hello'"],"old_keys_name":[],"old_keys_type":[],"old_keys_val":[]}
COMMIT XID: 15504
BEGIN CSN: 2015 first_lsn: 0/2816C20
{"table_name":"public.t1","op_type":"UPDATE","columns_name":["a","b","c"],"columns_type":["integer","integer","text"],"columns_val":["1","5","'hello'"],"old_keys_name":["a"],"old_keys_type":["integer"],"old_keys_val":["1"]}
COMMIT XID: 15505
```



```
BEGIN CSN: 2016 first_lsn: 0/2816D60
{"table_name":"public.t1","op_type":"DELETE","columns_name":[],"columns_type":[],"columns_val":
[],"old_keys_name":["a"],"old_keys_type":["integer"],"old_keys_val":["1"]}
COMMIT XID: 15506
```

5.3.16 Example: Parameters for Connecting to the Database in Different Scenarios

NOTE

In the example, **host:port** represents a node, where **host** indicates the name or IP address of the server where the database resides, and **port** indicates the port number of the server where the database resides.

DR

A customer has two database instances. Database instance A is the production database instance, and database instance B is the DR database instance. When the customer performs a DR switchover, database instance A is demoted to the DR database instance, and database instance B is promoted the production database instance. In this case, to avoid application restart or re-release caused by modifications on the configuration file, the customer can write database instances A and B to the connection string when initializing the configuration file. If the primary database instance cannot be connected, the driver attempts to connect to the DR database instance. For example, database instance A consists of *node1*, *node2*, and *node3*, and database instance B consists of *node4*, *node5*, and *node6*.

The URL can be configured as follows:

```
jdbc:gaussdb://node1,node2,node3,node4,node5,node6/database?priorityServers=3
```

If you want to connect to both the primary cluster and hosts in the primary cluster, set **targetServerType** to **master**. The URL can be configured as follows:

```
jdbc:gaussdb://node1,node2,node3,node4,node5,node6/database?
priorityServers=3&targetServerType=master
```

Load Balancing

A customer has a centralized database instance that consists of one primary node and two standby nodes, that is, *node1*, *node2*, and *node3*. *node1* is the primary node, and *node2* and *node3* are the standby nodes.

If the customer wants to evenly distribute the connections established on the same application to three nodes, the URL can be configured as follows:

```
jdbc:gaussdb://node1,node2,node3/database?loadBalanceHosts=true
```

CAUTION

When **loadBalanceHosts** is used, if the connection is established on the standby DN, write operations cannot be performed. If write operations are required, do not set this parameter.

Automatic Selection of the Primary Node

A customer has a centralized database instance that consists of one primary node and two standby nodes, that is, *node1*, *node2*, and *node3*. *node1* is the primary node, and *node2* and *node3* are the standby nodes.

If the customer requires that the application connection be established on the primary DN and a new primary node be automatically selected to establish the connection during the primary/standby switchover, configure the URL as follows:

```
jdbc:gaussdb://node1,node2,node3/database?targetServerType=master
```

Log Diagnosis

If a customer encounters slow data import or some errors that are difficult to analyze, the trace log function can be enabled for diagnosis. The URL can be configured as follows:

```
jdbc:gaussdb://node1/database?loggerLevel=trace
```

High Performance

A customer may execute the same SQL statement for multiple times with different input parameters. To improve the execution efficiency, the **prepareThreshold** parameter can be enabled to avoid repeatedly generating execution plans. The URL can be configured as follows:

```
jdbc:gaussdb://node1/database?prepareThreshold=5
```

A customer queries 10 million data records at a time. To prevent memory overflow caused by simultaneous return of the data records, the **defaultRowFetchSize** parameter can be used. The URL can be configured as follows:

```
jdbc:gaussdb://node1/database?defaultRowFetchSize=50000
```

A customer needs to insert 10 million data records in batches. To improve efficiency, the **batchMode** parameter can be used. The URL can be configured as follows:

```
jdbc:gaussdb://node1/database?batchMode=on
```

Case Conversion

In the Oracle database, metadata is stored in uppercase letters by default. In the GaussDB, metadata is stored in lowercase letters by default. Therefore, after the metadata is migrated from Oracle to GaussDB, the uppercase letters changes to lowercase letters. If the original service involves the processing of uppercase metadata, you can enable this parameter. However, you are advised to modify the service code instead of using this method to solve the problem. If you have to use this function, ensure that the metadata in the current database is in lowercase to avoid problems.

```
jdbc:gaussdb://node1/database?uppercaseAttributeName=true
```

The APIs involved in DatabaseMetaData can be directly called based on input parameters. The methods of calling the APIs involved in ResultSetMetaData are as follows:

```
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("select * from test_supper");
```



```
ResultSetMetaData rsmd = rs.getMetaData();
for (int i = 1; i <= rsmd.getColumnCount(); i++) {
    System.out.println(rsmd.getColumnLabel(i) + "    " + rsmd.getColumnName(i));
}
```

Application Lossless Transparent (ALT)

During planned maintenance (such as restarting database instances, performing a switchover, restarting nodes, and restarting DNs), JDBC and DNs are disconnected. You can enable the planned ALT feature to keep the connection. To configure the URL, see the following description. **gns** does not support floating IP address configuration.

Enable only the status notification.

```
jdbc:gaussdb://node1/database?enableALT=true&gns=ip1:port1,ip2:port2,ip3:port3
```

Enable the connection acceleration and quick disconnection functions.

```
jdbc:gaussdb://node1/database?enableALT=true&altLevel=C&gns=ip1:port1,ip2:port2,ip3:port3
```

Enable the connection acceleration, quick disconnection, and planned maintenance functions.

```
jdbc:gaussdb://node1/database?enableALT=true&altLevel=P&gns=ip1:port1,ip2:port2,ip3:port3
```

5.3.17 JDBC API Reference

This section describes common JDBC interfaces. For more interfaces, check JDK1.8 (software package) and JDBC 4.2.

5.3.17.1 java.sql.Connection

This section describes java.sql.Connection, the API for connecting to a database.

Table 5-9 Support status for java.sql.Connection

Method Name	Description	Return Type	throws	JDBC4 Support
abort(Executor executor)	Aborts an open connection.	void	SQLException	Yes
clearWarnings()	Clears all warnings reported for this connection object.	void	SQLException	Yes
close()	Releases the database and JDBC resources of this connection object immediately instead of waiting for them to be automatically released.	void	SQLException	Yes

Method Name	Description	Return Type	throws	JDBC4 Support
commit()	Makes all changes made since the last commit/rollback permanent and releases any database locks currently held by this connection object. This method should be used only when the autocommit mode is disabled.	void	SQLException	Yes
createArrayOf(String typeName, Object[] elements)	Creates an array object using this factory method.	Array	SQLException	Yes
createBlob()	Constructs an object that implements the BLOB interface. The returned object does not contain data initially. The setBinaryStream and setBytes methods of the BLOB interface can be used to add data to BLOBs.	Blob	SQLException	Yes
createClob()	Constructs an object that implements the CLOB interface. The returned object does not contain data initially. The setAsciiStream, setCharacterStream, and setString methods of the CLOB interface can be used to add data to CLOBs.	Clob	SQLException	Yes
createSQLXML()	Constructs an object that implements the SQLXML interface. The returned object does not contain data initially. You can use the createXmlStreamWriter object of the SQLXML interface and the setString method to add data to an SQLXML object.	SQLXML	SQLException	Yes
createStatement()	Creates a statement object that sends SQL statements to a database.	Statement	SQLException	Yes

Method Name	Description	Return Type	throws	JDBC4 Support
createStatement(int resultSetType, int resultSetConcurrency)	Creates a statement object that will generate a ResultSet object with a given type and concurrency.	Statement	SQLException	Yes
createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Creates a statement object that will generate a ResultSet object with a given type and concurrency.	Statement	SQLException	Yes
getAutoCommit()	Retrieves the current autocommit mode for this connection object.	boolean	SQLException	Yes
getCatalog()	Obtains the current directory name of this connection object.	String	SQLException	Yes
getClientInfo()	Returns a list containing the name and current value of each client information property supported by the driver. If the client information property has not been set and has no default value, the value of this attribute may be **NULL**.	Properties	SQLException	Yes
getClientInfo(String name)	Returns the value of the client information property specified by name. This method may return **NULL** if the specified client information property has not been set and there is no default value. This method also returns **NULL** if the driver does not support the specified client information property name.	String	SQLException	Yes
getHoldability()	Obtains the current holdability of the ResultSet object created using this connection object.	int	SQLException	Yes

Method Name	Description	Return Type	throws	JDBC4 Support
getMetaData()	Retrieves the DatabaseMetaData object that contains metadata about the database to which this connection object represents the connection. Metadata includes information about database tables, supported SQL syntax, stored procedures, and functions of the connection.	DatabaseMetaData	SQLException	Yes
getNetworkTimeout()	Obtains the number of milliseconds the driver will wait for a database request to complete. If the limit is exceeded, SQLException is thrown.	int	SQLException	Yes
getSchema()	Retrieves the current schema name of this connection object.	String	SQLException	Yes
getTransactionIsolation()	Retrieves the current transaction isolation level of this connection object.	int	SQLException	Yes
getTypeMap()	Retrieves the map object associated with this connection object. Unless the application adds an entry, the returned type mapping will be empty.	Map<String,Class<?>>	SQLException	Yes
getWarnings()	Retrieves the first warning reported by calls on this connection object. If there are multiple warnings, subsequent warnings are linked to the first warning and can be retrieved by calling the SQLWarning.getNextWarning method on the previously retrieved warning.	SQLWarning	SQLException	Yes
isClosed()	Checks whether this connection object has been closed. If a method has been called to close the connection, or if some major error occurs, the connection is closed. The value **true** is returned only when this method is called after the close ction.Close method is called.	boolean	SQLException	Yes

Method Name	Description	Return Type	throws	JDBC4 Support
isReadOnly()	Checks whether this connection object is in read-only mode.	boolean	SQLException	Yes
isValid(int timeout)	Returns **true** if the connection has not been closed and is still valid. The driver should commit a query to the connection, or use other mechanisms to affirmatively verify that the connection is still valid when this method is called.	Boolean	SQLException	Yes
nativeSQL(String sql)	Converts a given SQL statement to the native SQL syntax of the system. The driver can convert the JDBC SQL syntax to the native SQL syntax of its system before sending it. This method returns the native form of the statement that the driver should have sent.	String	SQLException	Yes
prepareCall(String sql)	Creates a CallableStatement object that calls a database stored procedure. The CallableStatement object provides methods for setting its IN and OUT parameters, as well as methods for executing calls to stored procedures.	CallableStatement	SQLException	Yes
prepareCall(String sql, int resultSetType, int resultSetConcurrency)	Creates a CallableStatement object. This object will generate a ResultSet object with the given type and concurrency. This method is the same as the preparation call method above, but it allows overwriting the default result set type and concurrency.	CallableStatement	SQLException	Yes
prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Creates a CallableStatement object. This object will generate a ResultSet object with the given type and concurrency. This method is the same as the preparation call method above, but it allows overwriting the default result set type, result set concurrency type, and holdability.	CallableStatement	SQLException	Yes

Method Name	Description	Return Type	throws	JDBC4 Support
prepareStatement(String sql)	Creates a prepared statement object for sending parameterized SQL statements to a database.	PreparedStatement	SQLException	Yes
prepareStatement(String sql, int autoGeneratedKeys)	Creates the default prepared statement object that can retrieve automatically generated keys. The given constant tells the driver whether it should make the automatically generated key available for retrieval. If the SQL statement is not an INSERT statement or an SQL statement that can return an automatically generated key, this parameter is ignored.	PreparedStatement	SQLException	Yes
prepareStatement(String sql, int[] columnIndexes)	Creates the default prepared statement object that can return an automatically generated key specified by a given array. This array contains the indexes of the columns in the target table that contain the automatically generated keys, which should be available. If the SQL statement is not an INSERT statement or an SQL statement that can return an automatically generated key, the driver ignores the array.	PreparedStatement	SQLException	Yes
prepareStatement(String sql, int resultSetType, int resultSetConcurrency)	Creates a prepared statement object that will generate a ResultSet object with a given type and concurrency. This method is the same as the prepared statement method above, but it allows overwriting the default result set type and concurrency.	PreparedStatement	SQLException	Yes
prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Creates a prepared statement object that will generate a ResultSet object with a given type, concurrency, and holdability.	PreparedStatement	SQLException	Yes

Method Name	Description	Return Type	throws	JDBC4 Support
prepareStatement(String sql, String[] columnNames)	Creates the default prepared statement object that can return an automatically generated key specified by a given array. This array contains the names of the columns in the target table that contain the automatically generated keys, which should be returned. If the SQL statement is not an INSERT statement or an SQL statement that can return an automatically generated key, the driver ignores the array.	PreparedStatement	SQLException	Yes
releaseSavepoint(Savepoint savepoint)	Deletes the specified savepoint and subsequent savepoint objects from the current transaction. Any reference to the savepoint after it is deleted will cause an SQL exception to be thrown.	void	SQLException	Yes
rollback()	Undoes all changes made in the current transaction and releases any database locks currently held by this connection object. This method should be used only when the autocommit mode is disabled.	void	SQLException	Yes
rollback(Savepoint savepoint)	Undoes all changes made after a given savepoint object is set. This method should be used only when the autocommit mode is disabled.	void	SQLException	Yes
setAutoCommit(boolean autoCommit)	Sets the autocommit mode of this connection to the given state. If a connection is in autocommit mode, all its SQL statements are executed and committed as a single transaction. Otherwise, its SQL statements are grouped into transactions that are terminated by calls to method commits or method rollbacks. By default, new connections are in autocommit mode.	void	SQLException	Yes

Method Name	Description	Return Type	throws	JDBC4 Support
setClientInfo(Properties properties)	Sets the client information property value of the connection. The properties object contains the names and values of the client information properties to be set. The client information property set contained in the attribute list replaces the current client information property set on the connection. If the property currently set on the connection is not in the property list, the property is cleared.	void	SQLClientInfoException	Yes
setClientInfo(String name,String value)	Sets the client information property value specified by name to that specified by value.	void	SQLClientInfoException	Yes
setHoldability(int holdability)	Changes the default holdability of the ResultSet object created with this connection object to the given holdability. The default holdability of the ResultSet object can be determined by calling DatabaseMetaData.getResultSetHoldability.	void	SQLException	Yes
setNetworkTimeout(Executor executor, int milliseconds)	Sets the maximum time that a connection or an object created from a connection waits for the database to reply to any request. If any request is still not answered, the waiting method returns SQLException, and the connection or object created from the connection is marked as closed. Any subsequent use of the object other than the Close, isCabled, or Connection.isValid method will result in SQLException.	void	SQLException	Yes
setReadOnly(boolean readOnly)	Sets this connection to read-only mode as a hint for the driver to enable database optimization.	void	SQLException	Yes

Method Name	Description	Return Type	throws	JDBC4 Support
setSavepoint()	Creates an unnamed savepoint in the current transaction and returns a new savepoint object representing it.	Savepoint	SQLException	Yes
setSavepoint(String name)	Creates a savepoint with the given name in the current transaction and returns a new savepoint object representing it.	Savepoint	SQLException	Yes
setSchema(String schema)	Sets the given schema name to access.	void	SQLException	Yes
setTransactionIsolation(int level)	Attempts to change the transaction isolation level of this connection object to a given level. Constants defined in the connection interface are possible transaction isolation levels.	void	SQLException	Yes
setTypeMap(Map<String,Class<?>> map)	Installs the given TypeMap object as the type mapping for this connection object. Type mappings will be used for SQL structured types and custom mappings for different types.	void	SQLException	Yes

NOTICE

1. The autocommit mode is used by default within the API. If you disable it by running **setAutoCommit(false)**, all the statements executed later will be packaged in explicit transactions, and you cannot execute statements that cannot be executed within transactions.
2. Fully-encrypted databases cannot use setClientInfo("send_token", null) to transmit keys or use setClientInfo("clear_token", null) to destroy keys.

5.3.17.2 java.sql.CallableStatement

This section describes **java.sql.CallableStatement**, the API for executing the stored procedure.

Table 5-10 Support status for java.sql.CallableStatement

Method Name	Return Type	JDBC4 Is Supported or Not
getArray(int parameterIndex)	Array	Yes
getBigDecimal(int parameterIndex)	BigDecimal	Yes
getBlob(int parameterIndex)	Blob	Yes
getBoolean(int parameterIndex)	Boolean	Yes
getByte(int parameterIndex)	byte	Yes
getBytes(int parameterIndex)	byte[]	Yes
getClob(int parameterIndex)	Clob	Yes
getDate(int parameterIndex)	Date	Yes
getDate(int parameterIndex, Calendar cal)	Date	Yes
getDouble(int parameterIndex)	double	Yes
getFloat(int parameterIndex)	float	Yes
getInt(int parameterIndex)	int	Yes
getLong(int parameterIndex)	long	Yes
getObject(int parameterIndex)	Object	Yes
getObject(int parameterIndex, Class<T> type)	Object	Yes
getShort(int parameterIndex)	short	Yes
getSQLXML(int parameterIndex)	SQLXML	Yes
getString(int parameterIndex)	String	Yes
getNString(int parameterIndex)	String	Yes
getTime(int parameterIndex)	Time	Yes
getTime(int parameterIndex, Calendar cal)	Time	Yes
getTimestamp(int parameterIndex)	Timestamp	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
getTimestamp(int parameterIndex, Calendar cal)	Timestamp	Yes
registerOutParameter(int parameterIndex, int type)	void	Yes
registerOutParameter(int parameterIndex, int sqlType, int type)	void	Yes
wasNull()	boolean	Yes

NOTE

- The batch operation of statements containing OUT parameter is not allowed.
- The following methods are inherited from java.sql.Statement: close, execute, executeQuery, executeUpdate, getConnection, getResultSet, getUpdateCount, isClosed, setMaxRows, and setFetchSize.
- The following methods are inherited from java.sql.PreparedStatement: addBatch, clearParameters, execute, executeQuery, executeUpdate, getMetaData, setBigDecimal, setBoolean, setByte, setBytes, setDate, setDouble, setFloat, setInt, setLong, setNull, setObject, setString, setTime, and setTimestamp.
- The **registerOutParameter(int parameterIndex, int sqlType, int type)** method is used only to register the composite data type.

5.3.17.3 java.sql.DatabaseMetaData

This section describes **java.sql.DatabaseMetaData**, the API for defining database objects.

Table 5-11 Support status for java.sql.DatabaseMetaData

Method Name	Return Type	JDBC4 Is Supported or Not
allProceduresAreCallable()	Boolean	Yes
allTablesAreSelectable()	Boolean	Yes
autoCommitFailureClosesAll-ResultSets()	Boolean	Yes
dataDefinitionCausesTran-sactionCommit()	Boolean	Yes
dataDefinitionIgnoredIn-Transactions()	Boolean	Yes
deletesAreDetected(int type)	Boolean	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
doesMaxRowSizeInclude-Blobs()	Boolean	Yes
generatedKeyAlwaysRe-turned()	Boolean	Yes
getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)	ResultSet	Yes
getCatalogs()	ResultSet	Yes
getCatalogSeparator()	String	Yes
getCatalogTerm()	String	Yes
getClientInfoProperties()	ResultSet	Yes
getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)	ResultSet	Yes
getConnection()	Connection	Yes
getCrossReference(String parentCatalog, String parentSchema, String parentTable, String foreignCatalog, String foreignSchema, String foreignTable)	ResultSet	Yes
getDefaultTransactionIsola-tion()	int	Yes
getExportedKeys(String catalog, String schema, String table)	ResultSet	Yes
getExtraNameCharacters()	String	Yes
getFunctionColumns(String catalog, String schemaPattern, String functionNamePattern, String columnNamePattern)	ResultSet	Yes
getFunctions(String catalog, String schemaPattern, String functionNamePattern)	ResultSet	Yes
getIdentifierQuoteString()	String	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
getImportedKeys(String catalog, String schema, String table)	ResultSet	Yes
getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)	ResultSet	Yes
getMaxBinaryLiteralLength()	int	Yes
getMaxCatalogName-Length()	int	Yes
getMaxCharLiteralLength()	int	Yes
getMaxColumnName-Length()	int	Yes
getMaxColumnsInGroupBy()	int	Yes
getMaxColumnsInIndex()	int	Yes
getMaxColumnsInOrderBy()	int	Yes
getMaxColumnsInSelect()	int	Yes
getMaxColumnsInTable()	int	Yes
getMaxConnections()	int	Yes
getMaxCursorNameLength()	int	Yes
getMaxIndexLength()	int	Yes
getMaxLogicalLobSize()	default long	Yes
getMaxProcedureName-Length()	int	Yes
getMaxRowSize()	int	Yes
getMaxSchemaName-Length()	int	Yes
getMaxStatementLength()	int	Yes
getMaxStatements()	int	Yes
getMaxTableNameLength()	int	Yes
getMaxTablesInSelect()	int	Yes
getMaxUserNameLength()	int	Yes
getNumericFunctions()	String	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
getPrimaryKeys(String catalog, String schema, String table)	ResultSet	Yes
getPartitionTablePrimary-Keys(String catalog, String schema, String table)	ResultSet	Yes
getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)	ResultSet	Yes
getProcedures(String catalog, String schemaPattern, String procedureNamePattern)	ResultSet	Yes
getProcedureTerm()	String	Yes
getSchemas()	ResultSet	Yes
getSchemas(String catalog, String schemaPattern)	ResultSet	Yes
getSchemaTerm()	String	Yes
getSearchStringEscape()	String	Yes
getSQLKeywords()	String	Yes
getSQLStateType()	int	Yes
getStringFunctions()	String	Yes
getSystemFunctions()	String	Yes
getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)	ResultSet	Yes
getTimeDateFunctions()	String	Yes
getTypeInfo()	ResultSet	Yes
getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)	ResultSet	Yes
getURL()	String	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
getVersionColumns(String catalog, String schema, String table)	ResultSet	Yes
insertsAreDetected(int type)	Boolean	Yes
locatorsUpdateCopy()	Boolean	Yes
othersDeletesAreVisible(int type)	Boolean	Yes
othersInsertsAreVisible(int type)	Boolean	Yes
othersUpdatesAreVisible(int type)	Boolean	Yes
ownDeletesAreVisible(int type)	Boolean	Yes
ownInsertsAreVisible(int type)	Boolean	Yes
ownUpdatesAreVisible(int type)	Boolean	Yes
storesLowerCaseIdentifiers()	Boolean	Yes
storesMixedCaseIdentifiers()	Boolean	Yes
storesUpperCaseIdentifiers()	Boolean	Yes
supportsBatchUpdates()	Boolean	Yes
supportsCatalogsInDataManipulation()	Boolean	Yes
supportsCatalogsInIndexDefinitions()	Boolean	Yes
supportsCatalogsInPrivilegeDefinitions()	Boolean	Yes
supportsCatalogsInProcedureCalls()	Boolean	Yes
supportsCatalogsInTableDefinitions()	Boolean	Yes
supportsCorrelatedSubqueries()	Boolean	Yes
supportsDataDefinitionAndDataManipulationTransactions()	Boolean	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
supportsDataManipulation-TransactionsOnly()	Boolean	Yes
supportsGetGeneratedKeys()	Boolean	Yes
supportsMixedCaseIdentifi-ers()	Boolean	Yes
supportsMultipleOpenRe-sults()	Boolean	Yes
supportsNamedParameters()	Boolean	Yes
supportsOpenCursorsAcross-Commit()	Boolean	Yes
supportsOpenCursorsAcross-Rollback()	Boolean	Yes
supportsOpenStatementsA-crossCommit()	Boolean	Yes
supportsOpenStatementsA-crossRollback()	Boolean	Yes
supportsPositionedDelete()	Boolean	Yes
supportsPositionedUpdate()	Boolean	Yes
supportsRefCursors()	Boolean	Yes
supportsResultSetConcurren-cy(int type, int concurrency)	Boolean	Yes
supportsResultSetType(int type)	Boolean	Yes
supportsSchemasInIndexDe-finitions()	Boolean	Yes
supportsSchemasInPrivilege-Definitions()	Boolean	Yes
supportsSchemasInProce-dureCalls()	Boolean	Yes
supportsSchemasInTableDe-finitions()	Boolean	Yes
supportsSelectForUpdate()	Boolean	Yes
supportsStatementPooling()	Boolean	Yes
supportsStoredFunctionsU-singCallSyntax()	Boolean	Yes
supportsStoredProcedures()	Boolean	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
supportsSubqueriesInComparisons()	Boolean	Yes
supportsSubqueriesInExists()	Boolean	Yes
supportsSubqueriesInIns()	Boolean	Yes
supportsSubqueriesInQuantifieds()	Boolean	Yes
supportsTransactionIsolationLevel(int level)	Boolean	Yes
supportsTransactions()	Boolean	Yes
supportsUnion()	Boolean	Yes
supportsUnionAll()	Boolean	Yes
updatesAreDetected(int type)	Boolean	Yes
getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)	ResultSet	Yes
getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)	ResultSet	Yes
getTableTypes()	ResultSet	Yes
getUserName()	String	Yes
isReadOnly()	Boolean	Yes
nullsAreSortedHigh()	Boolean	Yes
nullsAreSortedLow()	Boolean	Yes
nullsAreSortedAtStart()	Boolean	Yes
nullsAreSortedAtEnd()	Boolean	Yes
getDatabaseProductName()	String	Yes
getDatabaseProductVersion()	String	Yes
getDriverName()	String	Yes
getDriverVersion()	String	Yes
getDriverMajorVersion()	int	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
getDriverMinorVersion()	int	Yes
usesLocalFiles()	Boolean	Yes
usesLocalFilePerTable()	Boolean	Yes
supportsMixedCaseIdentifiers()	Boolean	Yes
storesUpperCaseIdentifiers()	Boolean	Yes
storesLowerCaseIdentifiers()	Boolean	Yes
supportsMixedCaseQuotedIdentifiers()	Boolean	Yes
storesUpperCaseQuotedIdentifiers()	Boolean	Yes
storesLowerCaseQuotedIdentifiers()	Boolean	Yes
storesMixedCaseQuotedIdentifiers()	Boolean	Yes
supportsAlterTableWithAddColumn()	Boolean	Yes
supportsAlterTableWithDropColumn()	Boolean	Yes
supportsColumnAliasing()	Boolean	Yes
nullPlusNonNullIsNull()	Boolean	Yes
supportsConvert()	Boolean	Yes
supportsConvert(int fromType, int toType)	Boolean	Yes
supportsTableCorrelationNames()	Boolean	Yes
supportsDifferentTableCorrelationNames()	Boolean	Yes
supportsExpressionsInOrderBy()	Boolean	Yes
supportsOrderByUnrelated()	Boolean	Yes
supportsGroupBy()	Boolean	Yes
supportsGroupByUnrelated()	Boolean	Yes
supportsGroupByBeyondSelect()	Boolean	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
supportsLikeEscapeClause()	Boolean	Yes
supportsMultipleResultSets()	Boolean	Yes
supportsMultipleTransactions()	Boolean	Yes
supportsNonNullableColumns()	Boolean	Yes
supportsMinimumSQLGrammar()	Boolean	Yes
supportsCoreSQLGrammar()	Boolean	Yes
supportsExtendedSQLGrammar()	Boolean	Yes
supportsANSI92EntryLevelSQL()	Boolean	Yes
supportsANSI92IntermediateSQL()	Boolean	Yes
supportsANSI92FullSQL()	Boolean	Yes
supportsIntegrityEnhancementFacility()	Boolean	Yes
supportsOuterJoins()	Boolean	Yes
supportsFullOuterJoins()	Boolean	Yes
supportsLimitedOuterJoins()	Boolean	Yes
isCatalogAtStart()	Boolean	Yes
supportsSchemasInDataManipulation()	Boolean	Yes
supportsSavepoints()	Boolean	Yes
supportsResultSetHoldability(int holdability)	Boolean	Yes
getResultSetHoldability()	int	Yes
getDatabaseMajorVersion()	int	Yes
getDatabaseMinorVersion()	int	Yes
getJDBCMajorVersion()	int	Yes
getJDBCMinorVersion()	int	Yes

NOTE

If the value of **uppercaseAttributeName** is **true**, the following methods convert the query result to uppercase letters. The conversion range is the same as that of the **toUpperCase** method in Java.

- public ResultSet getProcedures(String catalog, String schemaPattern, String procedureNamePattern)
- public ResultSet getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)
- public ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)
- public ResultSet getSchemas(String catalog, String schemaPattern)
- public ResultSet getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)
- public ResultSet getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)
- public ResultSet getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)
- public ResultSet getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)
- public ResultSet getPrimaryKeys(String catalog, String schema, String table)
- protected ResultSet getImportedExportedKeys(String primaryCatalog, String primarySchema, String primaryTable, String foreignCatalog, String foreignSchema, String foreignTable)
- public ResultSet getIndexInfo(String catalog, String schema, String tableName, boolean unique, boolean approximate)
- public ResultSet getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)
- public ResultSet getFunctions(String catalog, String schemaPattern, String functionNamePattern)

CAUTION

The getPartitionTablePrimaryKeys(String catalog, String schema, String table) method is used to obtain the primary key column of a partitioned table that contains global indexes. The following is an example:

```
PgDatabaseMetaData dbmd = (PgDatabaseMetaData)conn.getMetaData();
dbmd.getPartitionTablePrimaryKeys("catalogName", "schemaName", "tableName");
```

5.3.17.4 java.sql.Driver

This section describes **java.sql.Driver**, the database driver API.

Table 5-12 Support status for java.sql.Driver

Method Name	Return Type	JDBC4 Is Supported or Not
acceptsURL(String url)	Boolean	Yes
connect(String url, Properties info)	Connection	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
jdbcCompliant()	Boolean	Yes
getMajorVersion()	int	Yes
getMinorVersion()	int	Yes
getParentLogger()	Logger	Yes
getPropertyInfo(String url, Properties info)	DriverPropertyInfo[]	Yes

5.3.17.5 java.sql.PreparedStatement

This section describes **java.sql.PreparedStatement**, the API for preparing statements.

Table 5-13 Support status for java.sql.PreparedStatement

Method Name	Return Type	JDBC4 Is Supported or Not
clearParameters()	void	Yes
execute()	Boolean	Yes
executeQuery()	ResultSet	Yes
excuteUpdate()	int	Yes
executeLargeUpdate()	long	No
getMetaData()	ResultSetMetaData	Yes
getParameterMetaData()	ParameterMetaData	Yes
setArray(int parameterIndex, Array x)	void	Yes
setAsciiStream(int parameterIndex, InputStream x, int length)	void	Yes
setBinaryStream(int parameterIndex, InputStream x)	void	Yes
setBinaryStream(int parameterIndex, InputStream x, int length)	void	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
setBinaryStream(int parameterIndex, InputStream x, long length)	void	Yes
setBlob(int parameterIndex, InputStream inputStream)	void	Yes
setBlob(int parameterIndex, InputStream inputStream, long length)	void	Yes
setBlob(int parameterIndex, Blob x)	void	Yes
setCharacterStream(int parameterIndex, Reader reader)	void	Yes
setCharacterStream(int parameterIndex, Reader reader, int length)	void	Yes
setClob(int parameterIndex, Reader reader)	void	Yes
setClob(int parameterIndex, Reader reader, long length)	void	Yes
setClob(int parameterIndex, Clob x)	void	Yes
setDate(int parameterIndex, Date x, Calendar cal)	void	Yes
setNull(int parameterIndex, int sqlType)	void	Yes
setNull(int parameterIndex, int sqlType, String typeName)	void	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
setObject(int parameterIndex, Object x)	void	Yes
setObject(int parameterIndex, Object x, int targetSqlType)	void	Yes
setObject(int parameterIndex, Object x, int targetSqlType, int scaleOrLength)	void	Yes
setSQLXML(int parameterIndex, SQLXML xmlObject)	void	Yes
setTime(int parameterIndex, Time x)	void	Yes
setTime(int parameterIndex, Time x, Calendar cal)	void	Yes
setTimestamp(int parameterIndex, Timestamp x)	void	Yes
setTimestamp(int parameterIndex, Timestamp x, Calendar cal)	void	Yes
setUnicodeStream(int parameterIndex, InputStream x, int length)	void	Yes
setURL(int parameterIndex, URL x)	void	Yes
setBoolean(int parameterIndex, boolean x)	void	Yes
setBigDecimal(int parameterIndex, BigDecimal x)	void	Yes
setByte(int parameterIndex, byte x)	void	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
setBytes(int parameterIndex, byte[] x)	void	Yes
setDate(int parameterIndex, Date x)	void	Yes
setDouble(int parameterIndex, double x)	void	Yes
setFloat(int parameterIndex, float x)	void	Yes
setInt(int parameterIndex, int x)	void	Yes
setLong(int parameterIndex, long x)	void	Yes
setShort(int parameterIndex, short x)	void	Yes
setString(int parameterIndex, String x)	void	Yes
setNString(int parameterIndex, String x)	void	Yes
addBatch()	void	Yes
executeBatch()	int[]	Yes

NOTE

- Execute **addBatch()** and **execute()** only after running **clearBatch()**.
- Batch is not cleared by calling executeBatch(). Clear batch by explicitly calling clearBatch().
- After bounded variables of a batch are added, if you want to reuse these values, you do not need to use set*() again. Instead, add a batch.
- The following methods are inherited from java.sql.Statement: close, execute, executeQuery, executeUpdate, getConnection, getResultSet, getUpdateCount, isClosed, setMaxRows, setFetchSize, and enableStreamingResults.
- The executeLargeUpdate() method can only be used in JDBC 4.2 or later.

5.3.17.6 java.sql.ResultSet

This section describes **java.sql.ResultSet**, the API for execution result sets.

Table 5-14 Support status for java.sql.ResultSet

Method Name	Return Type	JDBC4 Is Supported or Not
absolute(int row)	Boolean	Yes
afterLast()	void	Yes
beforeFirst()	void	Yes
cancelRowUpdates()	void	Yes
clearWarnings()	void	Yes
close()	void	Yes
deleteRow()	void	Yes
findColumn(String columnLabel)	int	Yes
first()	Boolean	Yes
getArray(int columnIndex)	Array	Yes
getArray(String columnLabel)	Array	Yes
getAsciiStream(int columnIndex)	InputStream	Yes
getAsciiStream(String columnLabel)	InputStream	Yes
getBigDecimal(int columnIndex)	BigDecimal	Yes
getBigDecimal(String columnLabel)	BigDecimal	Yes
getBinaryStream(int columnIndex)	InputStream	Yes
getBinaryStream(String columnLabel)	InputStream	Yes
getBlob(int columnIndex)	Blob	Yes
getBlob(String columnLabel)	Blob	Yes
getBoolean(int columnIndex)	Boolean	Yes
getBoolean(String columnLabel)	Boolean	Yes
getByte(int columnIndex)	byte	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
getBytes(int columnIndex)	byte[]	Yes
getByte(String columnLabel)	byte	Yes
getBytes(String columnLabel)	byte[]	Yes
getCharacterStream(int columnIndex)	Reader	Yes
getCharacterStream(String columnLabel)	Reader	Yes
getClob(int columnIndex)	Clob	Yes
getClob(String columnLabel)	Clob	Yes
getConcurrency()	int	Yes
getCursorName()	String	Yes
getDate(int columnIndex)	Date	Yes
getDate(int columnIndex, Calendar cal)	Date	Yes
getDate(String columnLabel)	Date	Yes
getDate(String columnLabel, Calendar cal)	Date	Yes
getDouble(int columnIndex)	double	Yes
getDouble(String columnLabel)	double	Yes
getFetchDirection()	int	Yes
getFetchSize()	int	Yes
getFloat(int columnIndex)	float	Yes
getFloat(String columnLabel)	float	Yes
getInt(int columnIndex)	int	Yes
getInt(String columnLabel)	int	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
getLong(int columnIndex)	long	Yes
getLong(String columnLabel)	long	Yes
getMetaData()	ResultSetMetaData	Yes
getObject(int columnIndex)	Object	Yes
getObject(int columnIndex, Class<T> type)	<T> T	Yes
getObject(int columnIndex, Map<String,Class<?>> map)	Object	Yes
getObject(String columnLabel)	Object	Yes
getObject(String columnLabel, Class<T> type)	<T> T	Yes
getObject(String columnLabel, Map<String,Class<?>> map)	Object	Yes
getRow()	int	Yes
getShort(int columnIndex)	short	Yes
getShort(String columnLabel)	short	Yes
getSQLXML(int columnIndex)	SQLXML	Yes
getSQLXML(String columnLabel)	SQLXML	Yes
getStatement()	Statement	Yes
getString(int columnIndex)	String	Yes
getString(String columnLabel)	String	Yes
getNString(int columnIndex)	String	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
getNString(String columnLabel)	String	Yes
getTime(int columnIndex)	Time	Yes
getTime(int columnIndex, Calendar cal)	Time	Yes
getTime(String columnLabel)	Time	Yes
getTime(String columnLabel, Calendar cal)	Time	Yes
getTimestamp(int columnIndex)	Timestamp	Yes
getTimestamp(int columnIndex, Calendar cal)	Timestamp	Yes
getTimestamp(String columnLabel)	Timestamp	Yes
getTimestamp(String columnLabel, Calendar cal)	Timestamp	Yes
getType()	int	Yes
getWarnings()	SQLWarning	Yes
insertRow()	void	Yes
isAfterLast()	Boolean	Yes
isBeforeFirst()	Boolean	Yes
isClosed()	Boolean	Yes
isFirst()	Boolean	Yes
isLast()	Boolean	Yes
last()	Boolean	Yes
moveToCurrentRow()	void	Yes
moveToInsertRow()	void	Yes
next()	Boolean	Yes
previous()	Boolean	Yes
refreshRow()	void	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
relative(int rows)	Boolean	Yes
rowDeleted()	Boolean	Yes
rowInserted()	Boolean	Yes
rowUpdated()	Boolean	Yes
setFetchDirection(int direction)	void	Yes
setFetchSize(int rows)	void	Yes
updateArray(int columnIndex, Array x)	void	Yes
updateArray(String columnLabel, Array x)	void	Yes
updateAsciiStream(int columnIndex, InputStream x, int length)	void	Yes
updateAsciiStream(String columnLabel, InputStream x, int length)	void	Yes
updateBigDecimal(int columnIndex, BigDecimal x)	void	Yes
updateBigDecimal(String columnLabel, BigDecimal x)	void	Yes
updateBinaryStream(int columnIndex, InputStream x, int length)	void	Yes
updateBinaryStream(String columnLabel, InputStream x, int length)	void	Yes
updateBoolean(int columnIndex, boolean x)	void	Yes
updateBoolean(String columnLabel, boolean x)	void	Yes
updateByte(int columnIndex, byte x)	void	Yes
updateByte(String columnLabel, byte x)	void	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
updateBytes(int columnIndex, byte[] x)	void	Yes
updateBytes(String columnLabel, byte[] x)	void	Yes
updateCharacter-Stream(int columnIndex, Reader x, int length)	void	Yes
updateCharacter-Stream(String columnLabel, Reader reader, int length)	void	Yes
updateDate(int columnIndex, Date x)	void	Yes
updateDate(String columnLabel, Date x)	void	Yes
updateDouble(int columnIndex, double x)	void	Yes
updateDouble(String columnLabel, double x)	void	Yes
updateFloat(int columnIndex, float x)	void	Yes
updateFloat(String columnLabel, float x)	void	Yes
updateInt(int columnIndex, int x)	void	Yes
updateInt(String columnLabel, int x)	void	Yes
updateLong(int columnIndex, long x)	void	Yes
updateLong(String columnLabel, long x)	void	Yes
updateNull(int columnIndex)	void	Yes
updateNull(String columnLabel)	void	Yes
updateObject(int columnIndex, Object x)	void	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
updateObject(int columnIndex, Object x, int scaleOrLength)	void	Yes
updateObject(String columnLabel, Object x)	void	Yes
updateObject(String columnLabel, Object x, int scaleOrLength)	void	Yes
updateRow()	void	Yes
updateShort(int columnIndex, short x)	void	Yes
updateShort(String columnLabel, short x)	void	Yes
updateSQLXML(int columnIndex, SQLXML xmlObject)	void	Yes
updateSQLXML(String columnLabel, SQLXML xmlObject)	void	Yes
updateString(int columnIndex, String x)	void	Yes
updateString(String columnLabel, String x)	void	Yes
updateTime(int columnIndex, Time x)	void	Yes
updateTime(String columnLabel, Time x)	void	Yes
updateTimestamp(int columnIndex, Timestamp x)	void	Yes
updateTimestamp(String columnLabel, Timestamp x)	void	Yes
wasNull()	Boolean	Yes

NOTE

- One statement cannot have multiple open ResultSets.
- The cursor that is used for traversing the ResultSet cannot be open after being committed.

5.3.17.7 java.sql.ResultSetMetaData

This section describes **java.sql.ResultSetMetaData**, which provides details about ResultSet object information.

Table 5-15 Support status for java.sql.ResultSetMetaData

Method Name	Return Type	JDBC4 Is Supported or Not
getCatalogName(int column)	String	Yes
getColumnClassName(int column)	String	Yes
getColumnCount()	int	Yes
getColumnDisplaySize(int column)	int	Yes
getColumnLabel(int column)	String	Yes
getColumnName(int column)	String	Yes
getColumnType(int column)	int	Yes
getColumnTypeName(int column)	String	Yes
getPrecision(int column)	int	Yes
getScale(int column)	int	Yes
getSchemaName(int column)	String	Yes
getTableName(int column)	String	Yes
isAutoIncrement(int column)	Boolean	Yes
isCaseSensitive(int column)	Boolean	Yes
isCurrency(int column)	Boolean	Yes
isDefinitelyWritable(int column)	Boolean	Yes
isNullable(int column)	int	Yes
isReadOnly(int column)	Boolean	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
isSearchable(int column)	Boolean	Yes
isSigned(int column)	Boolean	Yes
isWritable(int column)	Boolean	Yes

NOTE

When **uppercaseAttributeName** is set to **true**, the following APIs convert the query result to uppercase letters. The conversion range is 26 English letters.

- public String getColumnName(int column)
- public String getColumnLabel(int column)

5.3.17.8 java.sql.Statement

This section describes **java.sql.Statement**, the interface for executing SQL statements.

Table 5-16 Support status for java.sql.Statement

Method Name	Return Type	JDBC4 Is Supported or Not
addBatch(String sql)	void	Yes
clearBatch()	void	Yes
clearWarnings()	void	Yes
close()	void	Yes
closeOnCompletion()	void	Yes
execute(String sql)	Boolean	Yes
execute(String sql, int autoGeneratedKeys)	Boolean	Yes
execute(String sql, int[] columnIndexes)	Boolean	Yes
execute(String sql, String[] columnNames)	Boolean	Yes
executeBatch()	Boolean	Yes
executeQuery(String sql)	ResultSet	Yes
executeUpdate(String sql)	int	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
executeUpdate(String sql, int autoGeneratedKeys)	int	Yes
executeUpdate(String sql, int[] columnIndexes)	int	Yes
executeUpdate(String sql, String[] columnNames)	int	Yes
getConnection()	Connection	Yes
getFetchDirection()	int	Yes
getFetchSize()	int	Yes
getGeneratedKeys()	ResultSet	Yes
getMaxFieldSize()	int	Yes
getMaxRows()	int	Yes
getMoreResults()	Boolean	Yes
getMoreResults(int current)	Boolean	Yes
getResultSet()	ResultSet	Yes
getResultSetConcurrency()	int	Yes
getResultSetHoldability()	int	Yes
getResultSetType()	int	Yes
getQueryTimeout()	int	Yes
getUpdateCount()	int	Yes
getWarnings()	SQLWarning	Yes
isClosed()	Boolean	Yes
isCloseOnCompletion()	Boolean	Yes
isPoolable()	Boolean	Yes
setCursorName(String name)	void	Yes
setEscapeProcessing(boolean enable)	void	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
setFetchDirection(int direction)	void	Yes
setMaxFieldSize(int max)	void	Yes
setMaxRows(int max)	void	Yes
setPoolable(boolean poolable)	void	Yes
setQueryTimeout(int seconds)	void	Yes
setFetchSize(int rows)	void	Yes
cancel()	void	Yes
executeLargeUpdate(String sql)	long	No
getLargeUpdateCount()	long	No
executeLargeBatch()	long	No
executeLargeUpdate(String sql, int autoGeneratedKeys)	long	No
executeLargeUpdate(String sql, int[] columnIndexes)	long	No
executeLargeUpdate(String sql, String[] columnNames)	long	No
enableStreamingResults()	void	Yes

NOTE

- Using setFetchSize can reduce the memory occupied by result sets on the client. Result sets are packaged into cursors and segmented for processing, which will increase the communication traffic between the database and the client, affecting performance.
- Database cursors are valid only within their transactions. If **setFetchSize** is set, set **setAutoCommit(false)** and commit transactions on the connection to flush service data to a database.
- The LargeUpdate method can only be used in JDBC 4.2 or later.
- enableStreamingResults() is a customized API for enabling streaming read. This API indirectly calls setFetchSize(Integer.MIN_VALUE). To enable the streaming read function, set **enableStreamingQuery** in the URL to **true** and call setFetchSize(Integer.MIN_VALUE) or enableStreamingResults(). Except that the streaming read function is enabled, the input parameter of **setFetchSize()** can only be a positive number or **0**.

5.3.17.9 javax.sql.ConnectionPoolDataSource

This section describes **javax.sql.ConnectionPoolDataSource**, the API for data source connection pools.

Table 5-17 Support status for javax.sql.ConnectionPoolDataSource

Method Name	Return Type	JDBC4 Is Supported or Not
getPooledConnection()	PooledConnection	Yes
getPooledConnection(String user,String password)	PooledConnection	Yes

5.3.17.10 javax.sql.DataSource

This section describes **javax.sql.DataSource**, the interface for data sources.

Table 5-18 Support status for javax.sql.DataSource

Method Name	Return Type	JDBC4 Is Supported or Not
getConneciton()	Connection	Yes
getConnection(String username,String password)	Connection	Yes
getLoginTimeout()	int	Yes
getLogWriter()	PrintWriter	Yes
setLoginTimeout(int seconds)	void	Yes
setLogWriter(PrintWriter out)	void	Yes

5.3.17.11 javax.sql.PooledConnection

This section describes **javax.sql.PooledConnection**, the connection API created by a connection pool.

Table 5-19 Support status for javax.sql.PooledConnection

Method Name	Return Type	JDBC4 Is Supported or Not
addConnectionEventListener (ConnectionEventListener listener)	void	Yes
close()	void	Yes
getConnection()	Connection	Yes
removeConnectionEventListener (ConnectionEventListener listener)	void	Yes

5.3.17.12 javax.naming.Context

This section describes **javax.naming.Context**, the context interface for connection configuration.

Table 5-20 Support status for javax.naming.Context

Method Name	Return Type	JDBC4 Is Supported or Not
bind(Name name, Object obj)	void	Yes
bind(String name, Object obj)	void	Yes
lookup(Name name)	Object	Yes
lookup(String name)	Object	Yes
rebind(Name name, Object obj)	void	Yes
rebind(String name, Object obj)	void	Yes
rename(Name oldName, Name newName)	void	Yes
rename(String oldName, String newName)	void	Yes

Method Name	Return Type	JDBC4 Is Supported or Not
unbind(Name name)	void	Yes
unbind(String name)	void	Yes

5.3.17.13 javax.naming.spi.InitialContextFactory

This section describes **javax.naming.spi.InitialContextFactory**, the initial context factory interface.

Table 5-21 Support status for javax.naming.spi.InitialContextFactory

Method Name	Description	Return Type	throws	JDBC4 Is Supported or Not
getInitialContext(Hashtable<?,?> environment)	Creates an initial context for starting name parsing. Special requirements for this context are provided by the environment.	Context	NamingException	Yes

5.3.17.14 CopyManager

CopyManager is an API class provided by the JDBC driver in GaussDB. It is used to import data to GaussDB in batches.

Inheritance Relationship of CopyManager

The CopyManager class is in the **com.huawei.gaussdb.jdbc.copy** package and inherits the java.lang.Object class. The declaration of the class is as follows:

```
public class CopyManager
extends Object
```

Constructor Method

public CopyManager(BaseConnection connection)

throws SQLException

Common Methods

Table 5-22 Common methods of CopyManager

Method	Return Value	Description	throws	JDBC4 Is Supported or Not
copyIn(String sql)	CopyIn	-	SQLException	Yes
copyIn(String sql, InputStream from)	long	Uses **COPY FROM STDIN** to quickly load data to tables in the database from InputStream.	SQLException,IOException	Yes
copyIn(String sql, InputStream from, int bufferSize)	long	Uses **COPY FROM STDIN** to quickly load data of a specified length to tables in the database from InputStream.	SQLException,IOException	Yes
copyIn(String sql, Reader from)	long	Uses **COPY FROM STDIN** to quickly load data to tables in the database from Reader.	SQLException,IOException	Yes
copyIn(String sql, Reader from, int bufferSize)	long	Uses **COPY FROM STDIN** to quickly load data of a specified length to tables in the database from Reader.	SQLException,IOException	Yes
copyOut(String sql)	CopyOut	-	SQLException	Yes
copyOut(String sql, OutputStream to)	long	Sends the result set of **COPY TO STDOUT** from the database to the OutputStream class.	SQLException,IOException	Yes
copyOut(String sql, Writer to)	long	Sends the result set of **COPY TO STDOUT** from the database to the Writer class.	SQLException,IOException	Yes

5.3.17.15 PGReplicationConnection

PGReplicationConnection is an API class provided by the JDBC driver in GaussDB. It is used to implement functions related to logical replication.

Inheritance Relationship of PGReplicationConnection

PGReplicationConnection is a logical replication interface. Its implementation class is PGReplicationConnectionImpl, which is in the **com.huawei.gaussdb.jdbc.replication** package. The declaration of the class is as follows:

```
public class PGReplicationConnection implements PGReplicationConnection
```

Constructor Method

public PGReplicationConnection(BaseConnection connection)

Common Methods

Table 5-23 Common methods of PGReplicationConnection

Return Value	Method	Description	Throws
ChainedCreateReplicationSlotBuilder	createReplication-Slot()	Creates a logical replication slot. Only LSN-based logical replication slots can be created. For details about how to create CSN-based logical replication slots, see the SQL function pg_create_logical_replication_slot for logical replication.	-
void	dropReplication-Slot(String slotName)	Deletes a logical replication slot.	SQLException,IOException
ChainedStreamBuilder	replicationStream()	Enables logical replication.	-

5.3.17.16 PGReplicationStream

PGReplicationStream is an API class provided by the GaussDB JDBC driver. It is used to operate logical replication streams.

Inheritance Relationship of PGReplicationStream

PGReplicationStream is a logical replication API. Its implementation class is V3PGReplicationStream, which is in the **com.huawei.gaussdb.jdbc.core.v3.replication** package. The declaration of the class is as follows:

```
public class V3PGReplicationStream implements PGReplicationStream
```

Constructor Method

public V3PGReplicationStream(CopyDual copyDual, LogSequenceNumber startLSN, long updateIntervalMs, ReplicationType replicationType)

Common Methods

Table 5-24 Common methods of PGReplicationStream

Return Value	Method	Description	throws
void	close()	Ends the logical replication and releases resources.	SQLException
void	forceUpdateStatus()	Forcibly sends the LSN status received, flushed, and applied last time to the backend.	SQLException
LogSequenceNumber	getLastAppliedLSN()	Obtains the LSN when the primary node replays logs last time.	-
LogSequenceNumber	getLastFlushedLSN()	Obtains the LSN flushed by the primary node last time, that is, the LSN pushed by the current logic decoding.	-
LogSequenceNumber	getLastReceiveLSN()	Obtains the last received LSN (for LSN-based replication slots) or CSN (for CSN-based replication slots).	-
boolean	isClosed()	Determines whether the replication stream is disabled.	-

Return Value	Method	Description	throws
ByteBuffer	read()	Reads the next WAL record from the backend. If the data cannot be read, this method blocks the I/O read.	SQLException
ByteBuffer	readPending()	Reads the next WAL record from the backend. If the data cannot be read, this method does not block the I/O read.	SQLException
void	setAppliedLSN(LogSequenceNumber applied)	Sets the applied LSN.	-
void	setFlushedLSN(LogSequenceNumber flushed)	Updates the LSN (for LSN-based replication slots) or CSN (for CSN-based replication slots), which is sent to the backend at the next update to update the LSN (for LSN-based replication slots) or CSN (for CSN-based replication slots) on the server.	-

5.3.17.17 ChainedStreamBuilder

ChainedStreamBuilder is an API class provided by the GaussDB JDBC driver. It is used to build replication streams.

Inheritance Relationship of ChainedStreamBuilder

ChainedStreamBuilder is a logical replication API. Its implementation class is ReplicationStreamBuilder, which is in the **com.huawei.gaussdb.jdbc.replication.fluent** package. The declaration of the class is as follows:

```
public class ReplicationStreamBuilder implements ChainedStreamBuilder
```

Constructor Method

public ReplicationStreamBuilder(final BaseConnection connection)

Common Methods

Table 5-25 Common methods of ReplicationStreamBuilder

Return Value	Method	Description	throws
ChainedLogicalStreamBuilder	logical()	Creates a logical replication stream.	-
ChainedPhysical-StreamBuilder	physical()	Creates a physical replication stream.	-

5.3.17.18 ChainedCommonStreamBuilder

ChainedCommonStreamBuilder is an API class provided by the GaussDB JDBC driver. It is used to specify common parameters for logical and physical replication.

Inheritance Relationship of ChainedCommonStreamBuilder

ChainedCommonStreamBuilder is an API for logical replication. The implementation abstract class is AbstractCreateSlotBuilder. The inheritance class is LogicalCreateSlotBuilder which is in the **com.huawei.gaussdb.jdbc.replication.fluent.logical** package. The declaration of the class is as follows:

```
public class LogicalCreateSlotBuilder
    extends AbstractCreateSlotBuilder<ChainedLogicalCreateSlotBuilder>
    implements ChainedLogicalCreateSlotBuilder
```

Constructor Method

public LogicalCreateSlotBuilder(BaseConnection connection)

Common Methods

Table 5-26 Common methods of LogicalCreateSlotBuilder

Return Value	Method	Description	throws
T	withSlotName(String slotName)	Specifies the name of a replication slot.	-
ChainedLogicalCreateSlotBuilder	withOutputPlugin(String outputPlugin)	Plug-in name. Currently, mppdb_decoding is supported.	-

Return Value	Method	Description	throws
void	make()	Creates a slot with the specified parameters in the database.	SQLException
ChainedLogicalCreateSlotBuilder	self()	Returns the implementation of ChainedLogicalCreateSlotBuilder.	-

5.3.17.19 PGobject

Table 5-27 Common PGobject Methods

Return Value	Method	Description	throws	JDBC4 Is Supported or Not
Object[]	getStruct()	Obtains the subtype names of the composite type and sorts them according to the creation sequence.	-	Yes
String	getValue()	Obtains the string value of the composite type.	-	Yes
String[]	getArrayValue()	Obtains the array values of the composite type and sorts the values according to the sequence of columns of the composite type.	-	Yes
Object[]	getAttributes()	Obtains the array values of the composite type. If the columns are of the table or array type, PgArray is returned. If the columns are of the composite type, PGobject is returned. For other types, a string value is returned.	SQLException	Yes

5.3.18 Common JDBC Parameters

targetServerType

Principle: If the value is **master**, it attempts to connect to the IP addresses configured in the string in sequence until the primary node in the cluster is connected.

If the value is **slave**, it attempts to connect to the IP addresses configured in the string in sequence until the standby node in the cluster is connected. The query statement is **select local_role, db_state from pg_stat_get_stream_replications()**.

Suggestion: You are advised to set this parameter to **master** for services with write operations to ensure that the primary node can be properly connected after a primary/standby switchover. However, if the standby node is not completely promoted to primary during the primary/standby switchover, the connection cannot be established. As a result, service statements cannot be executed.

hostRecheckSeconds

Principle: Specifies the period during which the DN list stored in JDBC remains trusted. Within this period, the DN list is directly read from the host addresses stored in JDBC. After that (or the primary node fails to be connected within the specified period), the node status in the DN list is updated and other IP addresses are connected.

Suggestion: The default value is **10s**. You are advised to adjust the value based on service requirements. This parameter is used together with the **targetServerType** parameter.

allowReadOnly

Principle: Checks whether the transaction access mode can be modified through **setReadOnly**. If the value is **true**, the transaction access mode can be modified. If the value is **false**, the transaction access mode cannot be modified through this API. To modify the transaction access mode, execute the statements: SET SESSION CHARACTERISTICS AS TRANSACTION + READ ONLY / READ WRITE.

Suggestion: The default value **true** is recommended.

fetchsize

Principle: After **fetchsize** is set to *n* and the database server executes a query, JDBC communicates with the server when the caller executes **resultset.next()**, fetches *n* pieces of data to the JDBC client, and returns the first piece of data to the caller. When the caller fetches the (*n*+1)th data record, the caller fetches data from the database server again.

Function: Prevents the database from transmitting all results to the client at a time, which exhausts the memory resources of the client.

Suggestion: You are advised to set this parameter based on the amount of data queried by services and the memory of the client. When setting **fetchsize**, disable automatic commit (set **autocommit** to **false**). Otherwise, the setting of **fetchsize** does not take effect.

defaultRowFetchSize

Function: The default value of **fetchsize** is **0**. Setting **defaultRowFetchSize** will change the default value of **fetchsize**.

batchMode

Function: This parameter specifies whether to connect the database in batch mode. The default value is **on**. After the function is enabled, the batch update performance is improved, and the return value is also batch updated. For example, if three data records are inserted in batches, the return value is **[3,0,0]** when the function is enabled, and the return value is **[1,1,1]** when the function is disabled.

Suggestion: If the service framework (such as hibernate) checks the return value during batch update, you can set this parameter to solve the problem.

loginTimeout

Function: Controls the time for establishing a connection with the database. The time includes connectTimeout and socketTimeoutInConnecting. If the time elapsed exceeds the threshold, the connection exits. The calculation formula is as follows: **loginTimeout** = (**connectTimeout** + Connection authentication time + Initialization statement execution time) x Number of nodes.

Suggestion: After this parameter is set, an asynchronous thread is started each time a connection is established. If there are a large number of connections, the pressure on the client may increase. If this parameter needs to be set, you are advised to set it to **max(connectTimeout, socketTimeoutInConnecting) x Number of nodes** in centralized deployment to prevent connection failures when the network is abnormal and the *n*th IP address is the IP address of the primary node. The default value is **0**.

NOTICE

- This parameter sets the time for attempting to connect to all IP addresses in a list. If this parameter is set to a small value, the subsequent IP addresses in the list may fail to be connected. For example, if three IP addresses are set, **loginTimeout** is set to **5s**, and it takes 5s to connect to the first two IP addresses, the third IP address cannot be connected. In a centralized environment, the last IP address is the IP address of the primary node. As a result, the automatic search for the primary node may fail.
- When any of the CPU, memory, and I/O load approaches 100%, the connection is slow, which may cause the connection time to exceed the threshold. You can locate the fault as follows:
  1. Log in to a physical machine with slow connections or use a management tool to query the resource load. You can run the **top** command to check the CPU usage, run the **free** command to check the memory usage, and run the **iostat** command to check the I/O load. In addition, you can check the monitoring logs in the CM Agent and the monitoring records on the database O&M platform.
  2. For peak load scenarios caused by a large number of slow queries in a short period of time, you can use the port specified by [*Port number of the database server* + 1] to query the pg_stat_activity view. For slow queries, you can use the system function **pg_terminate_backend(pid int)** to kill sessions.
  3. If service overloading exists for a long time (that is, there is no obvious slow query, or new queries still become slow after slow queries are killed), reduce the service load and increase database resources.

cancelSignalTimeout

Function: A cancel message may cause a block. This parameter controls connect timeout and socket timeout in a cancel command, in seconds. It is used to prevent timeout detection from being performed when the connection is canceled due to timeout.

Suggestion: The default value is **10s**. You are advised to adjust the value based on service requirements.

connectTimeout

Function: Controls the socket timeout threshold during connection setup. In this case, this timeout threshold is the time when the JDBC connects to the database through the socket, not the time when the connection object is returned. If the time elapsed exceeds the threshold, JDBC searches for the next IP address.

Suggestion: This parameter determines the maximum timeout interval for establishing a TCP connection on each node. If a network fault occurs on a node, it attempts to connect to the node until the time specified by **connectTimeout** elapses, and then attempts to connect to the next node. Considering the network jitter and delay, you are advised to set this parameter to **3s**.

socketTimeout

Function: Controls the timeout threshold of socket operations. If the time of executing service statements or reading data streams from the network exceeds the threshold (that is, when the statement execution time exceeds the specified threshold and no data is returned), the connection is interrupted.

Suggestion: This parameter specifies the maximum execution time of a single SQL statement. If the execution time of a single SQL statement exceeds the value of this parameter, an error is reported and the statement exits. You are advised to set this parameter based on service characteristics. If this parameter is not set, the default value **0** is used, indicating that the execution of SQL statement does not time out. If this parameter is not set, the client waits for a long time when the database process is abnormal. You are advised to set this parameter based on the SQL execution time acceptable to services.

socketTimeoutInConnecting

Function:Controls the socket operation timeout value during connection establishment. If the time of reading data streams from the network exceeds the threshold, it attempts to search for the next node for connection.

Suggestion: This parameter affects only the socket timeout value during the connection establishment. If this parameter is not set, the default value **5s** is used.

autosave

Function: If the value is **always**, you can set a savepoint before each statement in a transaction. If an error is reported during statement execution in a transaction, the system returns to the latest savepoint. In this way, subsequent statements in the transaction can be properly executed and committed. If the value is **conservative**, a savepoint is set for each query. However, the system rolls back and retries only when there is an invalid statement. If the value is **never**, there is no savepoint.

Suggestion: You are advised not to set this parameter to avoid severe performance deterioration.

currentSchema

Function: Specifies the schema of the current connection. If this parameter is not set, the default schema is the username used for the connection.

Suggestion: You are advised to set this parameter to the schema where the service data is located. If the schema name contains special characters except letters, digits, and underscores (_), you are advised to enclose the schema name in quotation marks. Note that the schema name is case sensitive after quotation marks are added. If multiple schemas need to be configured, separate them with commas (,). Schemas containing special characters also need to be enclosed in quotation marks.

prepareThreshold

Function: The default value is **5**. If an SQL statement is executed for multiple consecutive times in a session and the number of execution times specified by

prepareThreshold is reached, JDBC does not send the PARSE command to the SQL statement but caches the SQL statement to improve the execution speed.

Suggestion: The default value is **5**. Adjust the value based on service requirements.

preparedStatementCacheQueries

Function: Specifies the number of queries cached in each connection. The default value is **256**. If more than 256 different queries are used in the **prepareStatement()** call, the least recently used query cache will be discarded.

Suggestion: The default value is **256**. Adjust the value based on service requirements. This parameter is used together with **prepareThreshold**.

blobMode

Function: Sets the **setBinaryStream** method to assign values to different types of data. The value **on** indicates that values are assigned to BLOB data. The value **off** indicates that values are assigned to bytea data. The default value is **on**. For example, you can assign values to parameters in the preparestatement and callablestatement objects.

Suggestion: The default value is **on**.

setAutocommit

Function: If the value is **true**, a transaction is automatically started when each statement is executed. After the execution is complete, the transaction is automatically committed. That is, each statement is a transaction. If the value is **false**, a transaction is automatically started. However, you need to manually commit the transaction.

Suggestion: Adjust the value based on service characteristics. If autocommit needs to be disabled for performance or other purposes, the application must ensure that transactions can be committed. For example, explicitly commit translations after specifying service SQL statements. Particularly, ensure that all transactions are committed before the client exits.

5.3.19 Troubleshooting

5.3.19.1 Incorrect batchMode Settings

Symptom

Set the URL parameters **batchMode** to **on** and **reWriteBatchedInserts** to **true**. After JDBC is used to insert data in batches, a message is displayed, indicating that the number of binding parameters is inconsistent with the number of parameters required by the statement.

```
bind message supplies * parameters, but prepared statement "" requires *
```

Example 1:

```
// conn is a created connection object. The URL parameters for creating the connection contain &batchMode=on&reWriteBatchedInserts=true.
```



```
// Bind parameters in batches and then execute the statement. The number of bound parameters does not match the number of columns in the rewritten INSERT statement. As a result, an exception is reported.
// java.sql.BatchUpdateException: bind message supplies 3 parameters, but prepared statement "" requires 6
PreparedStatement stmt = conn.prepareStatement("insert into test_tbl values (?, ?, ?)");

stmt.setInt(1, 1);
stmt.setString(2, "aaa");
stmt.setString(3, "bbbb");
stmt.addBatch();

stmt.setInt(1, 1);
stmt.setString(2, "aaa");
stmt.setString(3, "bbbb");
stmt.addBatch();

stmt.setInt(1, 1);
stmt.setString(2, "aaa");
stmt.setString(3, "bbbb");
stmt.addBatch();

stmt.executeBatch();
```

Cause Analysis

When **reWriteBatchedInserts** is set to **true**, the batch statement combines multiple SQL statements into one. As a result, the number of reserved parameter columns in the statement changes. If **batchMode** is set to **on**, parameters are bound based on the SQL statements before combination. As a result, the number of bound parameters is inconsistent with the number of parameters required by the statement.

Solution

If **reWriteBatchedInserts** is set to **true**, set **batchMode** to **off**.

5.3.20 Mapping for JDBC Data Types

The relationships among data types, Java variable types, and JDBC type indexes are as follows (A: Oracle-compatible; B: MySQL-compatible; M: M-compatible).

Compatibility Mode	Gauss Kernel Data Type	Java Variable Type	JDBC Type Index
A/B	oid	java.lang.Long	java.sql.Types.BIGINT
A/B/M	numeric	java.math.BigDecimal	java.sql.Types.NUMERIC
A/B/M	tinyint	java.lang.Integer	java.sql.Types.TINYINT
A/B/M	smallint	java.lang.Integer	java.sql.Types.SMALLINT
A/B/M	bigint	java.lang.Long	java.sql.Types.BIGINT
A/B/M	float4	java.lang.Float	java.sql.Types.REAL
A/B/M	float8	java.lang.Double	java.sql.Types.DOUBLE
A/B/M	char	java.lang.String	java.sql.Types.CHAR

Compatibility Mode	Gauss Kernel Data Type	Java Variable Type	JDBC Type Index
A/B	character	java.lang.String	java.sql.Types.CHAR
A/B	bpchar	java.lang.String	java.sql.Types.CHAR
A/B	character varying	java.lang.String	java.sql.Types.VARCHAR
A/B/M	varchar	java.lang.String	java.sql.Types.VARCHAR
A/B/M	text	java.lang.String	java.sql.Types.VARCHAR
A/B	name	java.lang.String	java.sql.Types.VARCHAR
A/B	bytea	byte[]	java.sql.Types.BINARY
A/B/M	blob	java.sql.Blob	java.sql.Types.BLOB
A/B	clob	java.sql.Clob	java.sql.Types.CLOB
A/B	boolean	java.lang.Boolean	java.sql.Types.BIT
B/M	date	java.sql.Date	java.sql.Types.DATE
A/B/M	time	java.sql.Time	java.sql.Types.TIME
A/B	timetz	java.sql.Time	java.sql.Types.TIME
A/B/M	timestamp	java.sql.Timestamp	java.sql.Types.TIMESTAMP
A/B	smalldatetime	java.sql.Timestamp	java.sql.Types.TIMESTAMP
A/B	timestamptz	java.sql.Timestamp	java.sql.Types.TIMESTAMP
A/B	refcursor	java.sql.ResultSet	java.sql.Types.REF_CURSOR java.sql.Types.OTHER -10
M	boolean	java.lang.Integer	java.sql.Types.TINYINT
M	tinyblob	java.sql.Blob	java.sql.Types.BLOB
M	mediumblob	java.sql.Blob	java.sql.Types.BLOB
M	longblob	java.sql.Blob	java.sql.Types.BLOB
M	tinytext	java.lang.String	java.sql.Types.VARCHAR
M	mediumtext	java.lang.String	java.sql.Types.VARCHAR
M	longtext	java.lang.String	java.sql.Types.VARCHAR
M	binary	byte[]	java.sql.Types.BINARY

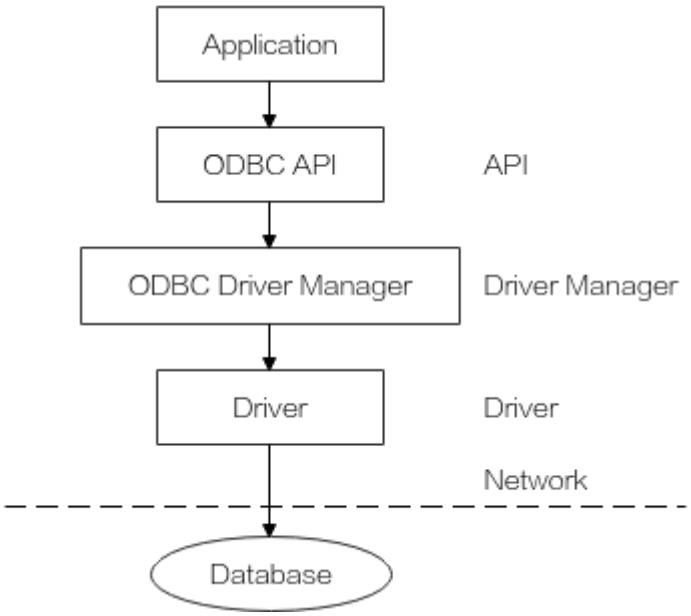
Compatibility Mode	Gauss Kernel Data Type	Java Variable Type	JDBC Type Index
M	varbinary	byte[]	java.sql.Types.BINARY
M	tinyint unsigned	java.lang.Integer	java.sql.Types.INTEGER
M	smallint unsigned	java.lang.Integer	java.sql.Types.INTEGER
M	mediumint unsigned	java.lang.Integer	java.sql.Types.INTEGER
M	integer unsigned	java.lang.Long	java.sql.Types.BIGINT
M	bigint unsigned	java.math.BigInteger	java.sql.Types.BIGINT
M	mediumint	java.lang.Integer	java.sql.Types.INTEGER
M	integer	java.lang.Integer	java.sql.Types.INTEGER
M	decimal	java.math.BigDecimal	java.sql.Types.NUMERIC
M	dec	java.math.BigDecimal	java.sql.Types.NUMERIC
M	real	java.lang.Double	java.sql.Types.DOUBLE
M	datetime	java.sql.Timestamp	java.sql.Types.TIMESTAMP
M	year	java.sql.Date	java.sql.Types.DATE
M	bit	byte[]	java.sql.Types.BIT

5.4 Development Based on ODBC

Open Database Connectivity (ODBC) is a Microsoft API for accessing databases based on the X/OPEN CLI. Applications interact with the database through the APIs provided by ODBC, which enhances their portability, scalability, and maintainability.

Figure 5-2 shows the system structure of ODBC.

Figure 5-2 ODBC system structure



GaussDB supports ODBC in the following environments.

Table 5-28 OSs Supported by ODBC

OS	Platform
EulerOS V2.0SP5	x86_64
EulerOS V2.0SP9	Arm64
EulerOS V2.0SP10	x86_64
EulerOS V2.0SP10	Arm64
Windows 7	x86_32
Windows 7	x86_64
Windows Server 2008	x86_32
Windows Server 2008	x86_64
Kylin V10	x86_64
Kylin V10	Arm64
UnionTech V20	x86_64
UnionTech V20	Arm64
Huawei Cloud EulerOS 2.0	x86_64
Huawei Cloud EulerOS 2.0	Arm64

The ODBC Driver Manager running on Unix or Linux can be unixODBC or iODBC. unixODBC-2.3.7 is used as the component for connecting to the database.

Windows has a native ODBC Driver Manager. You can locate **Data Sources (ODBC)** by choosing **Control Panel** > **Administrative Tools**.

NOTE

The current database ODBC driver is based on an open-source version and may be incompatible with data types tinyint, smalldatetime, nvarchar, and nvarchar2.

ODBC Constraints

- ODBC does not support read on the standby node.
- ODBC does not support user-defined types and does not support user-defined parameters in stored procedures.
- ODBC does not support DR switchover.
- When the **proc_outparam_override** parameter is enabled for the database, ODBC cannot properly call the stored procedure that contains the **out** parameter.

5.4.1 ODBC Packages, Dependent Libraries, and Header Files

ODBC Packages for the Linux OS

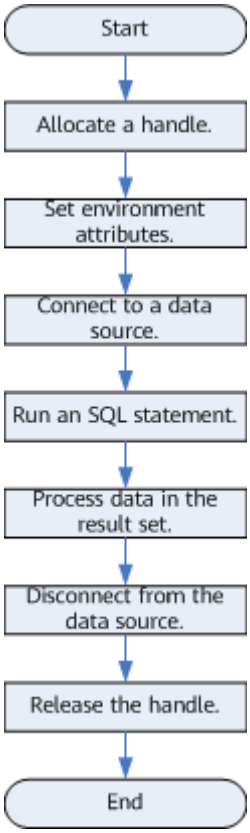
The package is obtained from the release package. The package name is **GaussDB-Kernel**_*Database version number*_*OS version number*_**64bit_Odbc.tar.gz**. In the Linux OS, header files (including **sql.h** and **sqlext.h**) and the library (**libodbc.so**) are required in application development. These header files and library can be obtained from the unixODBC-2.3.7 installation package.

ODBC Packages for the Windows OS

The package is obtained from the release package. The package name is **GaussDB-Kernel**_*Database version number*_**Windows_X86_Odbc.tar.gz** (32-bit) or **GaussDB-Kernel**_*Database version number*_**Windows_X64_Odbc.tar.gz** (64-bit). In the Windows OS, the required header files and library files are system-resident.

5.4.2 Development Process

Figure 5-3 ODBC-based application development process



APIs Involved in the Development Process

Table 5-29 API description

Function	API
Allocate a handle	**SQLAllocHandle** is a generic function for allocating a handle. It can replace the following functions: • **SQLAllocEnv**: allocates an environment handle. • **SQLAllocConnect**: allocates a connection handle. • **SQLAllocStmt**: allocates a statement handle.
Set environment attributes	**SQLSetEnvAttr**
Set connection attributes	**SQLSetConnectAttr**
Set statement attributes	**SQLSetStmtAttr**
Connect to a data source	**SQLConnect**

Function	API
Bind a buffer to a column in the result set	**SQLBindCol**
Bind the parameter marker of an SQL statement to a buffer	**SQLBindParameter**
Return the error message of the last operation	**SQLGetDiagRec**
Prepare an SQL statement for execution	**SQLPrepare**
Run a prepared SQL statement	**SQLExecute**
Run an SQL statement directly	**SQLExecDirect**
Fetch the next row (or rows) from the result set	**SQLFetch**
Return data in a column of the result set	**SQLGetData**
Get the column information from a result set	**SQLColAttribute**
Disconnect from a data source	**SQLDisconnect**
Release a handle	**SQLFreeHandle** is a generic function for releasing a handle. It can replace the following functions: • **SQLFreeEnv**: releases an environment handle. • **SQLFreeConnect**: releases a connection handle. • **SQLFreeStmt**: releases a statement handle.

NOTE

- ODBC connects applications to the database and delivers the SQL statements sent by an application to the database. It does not parse the SQL syntax. Therefore, when confidential information is written into the SQL statement sent by an application, the confidential information is exposed in the driver log.
- If an execution request (not in a transaction block) received in the database contains multiple statements, the request is packed into a transaction. If one of the statements fails, the entire request will be rolled back.

5.4.3 Configuring a Data Source in the Linux OS

The ODBC DRIVER (**gsqlodbcw.so**) provided by GaussDB can be used after it has been configured in a data source. To configure a data source, you must configure

the **odbc.ini** and **odbcinst.ini** files on the server. The two files are generated during the unixODBC compilation and installation, and are saved in the **/usr/local/etc** directory by default.

Procedure

Step 1 Obtain the source code package of unixODBC by clicking the following link:

Download address: **https://www.unixodbc.org/unixODBC-2.3.7.tar.gz**.

After the download, verify the integrity based on the integrity verification algorithm provided by the community. Download **https://www.unixodbc.org/unixODBC-2.3.7.tar.gz.md5**, view the MD5 value, and check whether the MD5 value is the same as that in the source code package.

Step 2 Install unixODBC. It does not matter if unixODBC of another version has been installed.

For example, to install unixODBC-2.3.7, run the commands below.

```
tar zxvf unixODBC-2.3.7.tar.gz
cd unixODBC-2.3.7

./configure --enable-gui=no # To perform compilation on an Arm server, add the configure parameter --build=aarch64-unknown-linux-gnu.
make
# The installation may require root permissions.
make install
```

NOTE

- Currently, unixODBC-2.2.1 is not supported.
- By default, it is installed in the **/usr/local** directory. The data source file is generated in the **/usr/local/etc** directory, and the library file is generated in the **/usr/local/lib** directory.
- You can compile unixODBC with the **--enable-fastvalidate=yes** option to achieve higher performance. However, this option may cause an application that passes an invalid handle to the ODBC API to fail instead of returning an SQL_INVALID_HANDLE error.

Step 3 Replace the GaussDB client driver.

Decompress **GaussDB-Kernel**_*Database version number*_*OS version number*_**64bit_Odbc.tar.gz**. After the decompression, the **lib** and **odbc** folders are generated. The **odbc** folder contains another **lib** folder. Copy all dynamic libraries in the **/lib** and **/odbc/lib** folders to the **/usr/local/lib** directory.

Step 4 Configure a data source.

1. Configure the ODBC driver file.

   Add the following content to the **/usr/local/etc/odbcinst.ini** file:

   ```
   [GaussMPP]
   Driver64=/usr/local/lib/gsqlodbcw.so
   setup=/usr/local/lib/gsqlodbcw.so
   ```

   For descriptions of the parameters in the **odbcinst.ini** file, see **Table 5-30**.

Table 5-30 odbcinst.ini configuration parameters

Parameter	Description	Example
[DriverName]	Driver name, corresponding to Driver in DSN.	[DRIVER_N]
Driver64	Path of the dynamic driver library.	Driver64=/usr/local/lib/gsqlodbcw.so
setup	Driver installation path, which is the same as the dynamic library path in Driver64.	setup=/usr/local/lib/gsqlodbcw.so

2. Configure the data source file.

   Add the following content to the **/usr/local/etc/odbc.ini** file:

```
[MPPODBC]
Driver=GaussMPP
Servername=127.0.0.1 # Database server IP address
Database=db1  # Database name
Username=omm  # Database username
Password=  # Database user password
Port=8000  # Database listening port
Sslmode = allow
```

   For descriptions of the parameters in the **odbc.ini** file, see **Table 5-31**.

Table 5-31 odbc.ini configuration parameters

Parameter	Description	Example
[DSN]	Data source name.	[MPPODBC]
Driver	Driver name, corresponding to DriverName in **odbcinst.ini**.	Driver = DRIVER_N
Servername	Server IP address. Multiple IP addresses can be configured. Both IPv4 and IPv6 are supported.	Servername=127.0.0.1
Database	Name of the database to connect.	Database=db1
Username	Database username.	Username = omm

Parameter	Description	Example
Password	Database user password. **NOTE** After a user establishes a connection, the ODBC driver automatically clears their password stored in memory. However, if this parameter is configured, unixODBC will cache data source files, which may cause the password to be stored in the memory for a long time. When you connect to an application, you are advised to send your password through an API instead of writing it in a data source configuration file. After the connection has been established, immediately clear the memory segment where your password is stored. **CAUTION** The password in the configuration file must comply with the following HTTP rules: 1. Characters must comply with the URL encoding specifications. For example, the exclamation mark (!) must be written as %21, and the percent sign (%) must be written as %25. Therefore, pay attention to the characters. 2. A plus sign (+) will be replaced by a space.	Password=********
Port	Port number of the server.	Port = 8000
Sslmode	Specifies whether to enable SSL.	Sslmode = allow

Parameter	Description	Example
Debug	If this parameter is set to **1**, the **mylog** file of the gsqlODBC driver will be printed. The directory generated for storing logs is **/tmp/**. If this parameter is set to **0**, no directory is generated.	Debug = 1
UseServerSidePrepare	Specifies whether to enable the extended query protocol for the database. The value can be **0** or **1**. The default value is **1**, indicating that the extended query protocol is enabled.	UseServerSidePrepare = 1
UseBatchProtocol	Specifies whether to enable the batch query protocol. If it is enabled, DML performance can be improved. The value can be **0** or **1** (default). – If this parameter is set to **0**, the batch query protocol is disabled (mainly for communication with earlier database versions). – If this parameter is set to **1** and **support_batch_bind** is set to **on**, the batch query protocol is enabled.	UseBatchProtocol = 1
ForExtensionConnector	This parameter specifies whether the savepoint is sent. The default value is **1**. If the value is **0**, the savepoint is sent. If the value is **1**, the savepoint is not sent.	ForExtensionConnector = 1

Parameter	Description	Example
ConnectionExtraInfo	Specifies whether to display the driver deployment path and process owner in the GUC parameter **connection_info**. **NOTE** The default value is **0**. If this parameter is set to **1**, the ODBC driver reports the driver deployment path and process owner to the database and displays the information in the GUC parameter **connection_info**. In this case, you can query the information from **PG_STAT_ACTIVITY**.	ConnectionExtraInfo = 1
BoolsAsChar	If this parameter is set to **Yes**, the Boolean value is mapped to the SQL_CHAR type. If this parameter is not set, the value is mapped to the SQL_BIT type. The default value is **Yes**.	BoolsAsChar = Yes
RowVersioning	When an attempt is made to update a row of data, setting this parameter to **Yes** allows the application to detect whether the data has been modified by other users. The default value is **No**.	RowVersioning = Yes
ShowSystemTables	If the value is **Yes**, the driver regards the system table as a common SQL table by default. The default value is **No**.	ShowSystemTables = Yes

Parameter	Description	Example
MaxCacheQueries	Controls the number of precompiled statements cached for each connection. If this parameter is set to **0**, the precompiled statement cache pool is disabled on the client. If this parameter is set to a value greater than 4096, the value **4096** is used. If the number of executed statements exceeds the upper limit specified by **MaxCacheQueries**, the least recently used statements are eliminated. The default value is **0**.	MaxCacheQueries=128
MaxCacheSizeMiB	Controls the total size of precompiled statements cached for each connection. This parameter takes effect when the value of **MaxCacheQueries** is greater than 0. If the total size of cached statements is greater than the value of **MaxCacheSizeMiB**, the least recently used statements are eliminated. If this parameter is set to a value greater than 4096, the value **4096** is used. The unit is MB. The default value is **1**.	MaxCacheSizeMiB=10

Parameter	Description	Example
TcpUserTimeout	Specifies the maximum duration for which transmitted data can remain unacknowledged before the TCP connection is forcibly closed on an OS that supports the **TCP_USER_TIMEOUT** socket option. **0** indicates that the default value is used. Ignore this parameter for UDS connections. The unit is millisecond. The default value is **0**.	TcpUserTimeout=5000

Table 3 Sslmode options describes the valid values of **Sslmode**.

Table 5-32 Sslmode options

Sslmode	Whether SSL Encryption Is Enabled	Description
disable	No	SSL connection is not enabled.
allow	Possible	If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified.
prefer	Possible	If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified.
require	Yes	SSL connection is required and data is encrypted. However, authenticity of the database server will not be verified.
verify-ca	Yes	SSL connection is required and whether the database has a trusted certificate will be verified.
verify-full	Yes	SSL connection is required. In addition to the check scope specified by **verify-ca**, the system checks whether the name of the host where the database resides is the same as that in the certificate. GaussDB does not support this mode.

Step 5 (Optional) Contact the database administrator to generate an SSL certificate. This step and **Step 6** are required only when the server and client are connected in SSL mode. Skip the two steps if the non-SSL connection mode is used.

Step 6 (Optional) Contact the database administrator to replace an SSL certificate.

Step 7 Enable the SSL mode. For details, contact the database administrator.

Step 8 Configure the database server. For details, contact the database administrator.

Step 9 Configure the environment variables on the client.

```
vim ~/.bashrc
```

Add the following information to the configuration file:

```
export LD_LIBRARY_PATH=/usr/local/lib/:$LD_LIBRARY_PATH
export ODBCSYSINI=/usr/local/etc
export ODBCINI=/usr/local/etc/odbc.ini
```

Step 10 Run the following command to validate the addition:

```
source ~/.bashrc
```

----End

Verifying the Data Source Configuration

After the installation, the generated binary file is stored in the **/usr/bin** directory. You can run the **isql -v** *MPPODBC* command (*MPPODBC* is the data source name).

- If the following information is displayed, the configuration is correct and the connection succeeds:

```
+---------------------------------------+
| Connected!                            |
|                                       |
| sql-statement                         |
| help [tablename]                      |
| quit                                  |
|                                       |
+---------------------------------------+
```

- If error information is displayed, the configuration is incorrect. Check the configuration.

FAQs

- [UnixODBC][Driver Manager]Can't open lib 'xxx/xxx/gsqlodbcw.so':file not found.

  Possible causes:

  – The path configured in the **odbcinst.ini** file is incorrect.

  Run **ls** to check the path in the error information, ensuring that the **gsqlodbcw.so** file exists and you have execution permissions on it.

  – The dependent library of **gsqlodbcw.so** does not exist or is not in system environment variables.

  Run the **ldd** command to check the path in the error information. If the unixODBC library such as **libodbc.so.1** is missing, reconfigure unixODBC according to the procedure, ensure that the **lib** directory in the installation path is added to *LD_LIBRARY_PATH*. If other libraries do not exist, add the **lib** directory under the ODBC driver package to *LD_LIBRARY_PATH*. If other standard libraries are missing, install them.

- [UnixODBC]connect to server failed: no such file or directory

  Possible causes:

  – An incorrect or unreachable database IP address or port was configured.

    Check the **Servername** and **Port** configuration items in data sources.

  – Server monitoring is improper.

    If **Servername** and **Port** are correctly configured, ensure the proper network adapter and port are monitored by following the database server configurations in the procedure in this section.

  – Firewall and network gatekeeper settings are improper.

    Check firewall settings, and ensure that the database communication port is trusted.

    Check to ensure that network gatekeeper settings are proper (if any).

- [unixODBC]The password-stored method is not supported.

  Possible causes:

  The **sslmode** configuration item is not configured in the data sources.

  Solution:

  Set the configuration item to **allow** or a higher level. For details, see **Table 5-32**.

- Server common name "xxxx" does not match host name "xxxxx"

  Possible causes:

  When **verify-full** is used for SSL encryption, the driver checks whether the host name in certificates is the same as the actual one.

  Solution:

  To solve this problem, use **verify-ca** to stop checking host names, or generate a set of CA certificates containing the actual host names.

- Driver's SQLAllocHandle on SQL_HANDLE_DBC failed

  Possible causes:

  The executable file (such as the **isql** of unixODBC) and the database driver (**gsqlodbcw.so**) depend on different library versions of ODBC, such as **libodbc.so.1** and **libodbc.so.2**. You can verify this problem by using the following method:

  ```
  ldd `which isql` | grep odbc
  ldd gsqlodbcw.so | grep odbc
  ```

  If the suffix digits of the outputs **libodbc.so** are different or indicate different physical disk files, this problem exists. Both **isql** and **gsqlodbcw.so** require **libodbc.so** to be loaded. If they load different physical files, two sets of function lists with the same name are generated in a visible domain (the **libodbc.so.**** function export lists of unixODBC are the same). This results in conflicts and the database driver cannot be loaded.

  Solution:

  Uninstall the unnecessary unixODBC, such as **libodbc.so.2**, and create a soft link with the same name and the .so.2 suffix for the remaining **libodbc.so.1** library.

- FATAL: Forbid remote connection with trust method!

  For security purposes, the primary database node forbids access from other nodes in the database without authentication.

To access the primary database node from inside the database, deploy the ODBC program on the host where the primary database node is located and set the server address to **127.0.0.1**. It is recommended that the service system be deployed outside the database. Otherwise, the database performance may be affected.

- [unixODBC][Driver Manager]Invalid attribute value

  The unixODBC version may not be the recommended one. You are advised to run the **odbcinst --version** command to check the unixODBC version in the environment.

- authentication method 10 not supported.

  If this error occurs on an open-source client, the cause may be:

  The database stores only the SHA-256 hash of the password, but the open-source client supports only MD5 hashes.

  NOTE

  - The database stores the hashes of user passwords instead of actual passwords.
  - If a password is updated or a user is created, both types of hashes will be stored, compatible with open-source authentication protocols.
  - An MD5 hash can only be generated using the original password, but the password cannot be obtained by reversing its SHA-256 hash. Passwords in the source version will only have SHA-256 hashes and not support MD5 authentication.
  - The MD5 encryption algorithm has lower security and poses security risks. Therefore, you are advised to use a more secure encryption algorithm.

  To solve this problem, you can update the user password (see **ALTER USER**) or create a user (see **CREATE USER**) having the same permissions as the faulty user.

- unsupported frontend protocol 3.51: server supports 1.0 to 3.0

  The database version is too early or the database is an open-source database. Use the driver of the required version to connect to the database.

- FATAL: GSS authentication method is not allowed because XXXX user password is not disabled.

  In **gs_hba.conf** of the target primary database node, the authentication mode is set to **gss** for authenticating the IP address of the current client. However, this authentication algorithm cannot authenticate clients. Change the authentication algorithm to **sha256** and try again. For details, see **Step 8**.

- isql: error while loading shared libraries:xxx

  The dynamic library does not exist in the environment. You need to install the corresponding library.

5.4.4 Configuring a Data Source in the Windows OS

Configure an ODBC data source using the ODBC data source manager preinstalled in the Windows OS.

Procedure

Step 1 Replace the GaussDB client driver.

Decompress the **GaussDB-Kernel**_*Database version number*_**Windows_X64_Odbc.tar.gz** (64-bit) driver package or **GaussDB-**

Kernel_*Database version number*_**Windows_X86_Odbc.tar.gz** (32-bit) driver package, and click **gsqlodbc.exe** to install the driver.

Step 2 Open the driver manager.

When configuring the data source, use the ODBC driver manager corresponding to the ODBC version. If the 64-bit ODBC driver is used, the 64-bit ODBC driver manager must be used. Assume that the OS is installed on drive C (if the OS is installed on another drive, change the path accordingly):

- If you want to use 32-bit ODBC driver manager in a 64-bit OS, open **C:\Windows\SysWOW64\odbcad32.exe**. Do not choose **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)**.

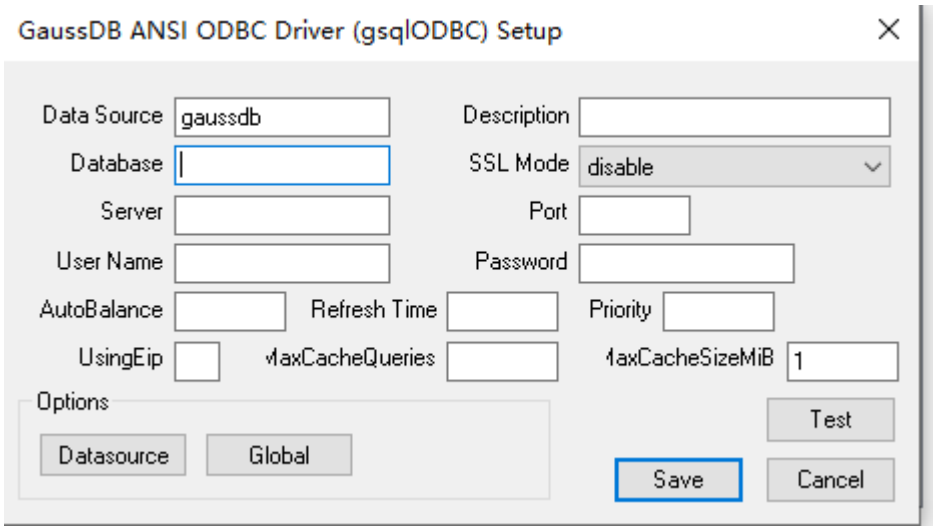
  NOTE

  WoW64 is short for Windows 32-bit on Windows 64-bit. **C:\Windows\SysWOW64** stores the 32-bit environment on a 64-bit system. **C:\Windows\System32** stores the environment consistent with the current OS. For technical details, see Windows technical documents.

- For a 32-bit OS, open **C:\Windows\System32\odbcad32.exe** or choose **Computer** > **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)** to open Driver Manager.
- For a 64-bit OS, choose **Control Panel** > **Administrative Tools** > **Data Sources (ODBC)** to enable driver management.

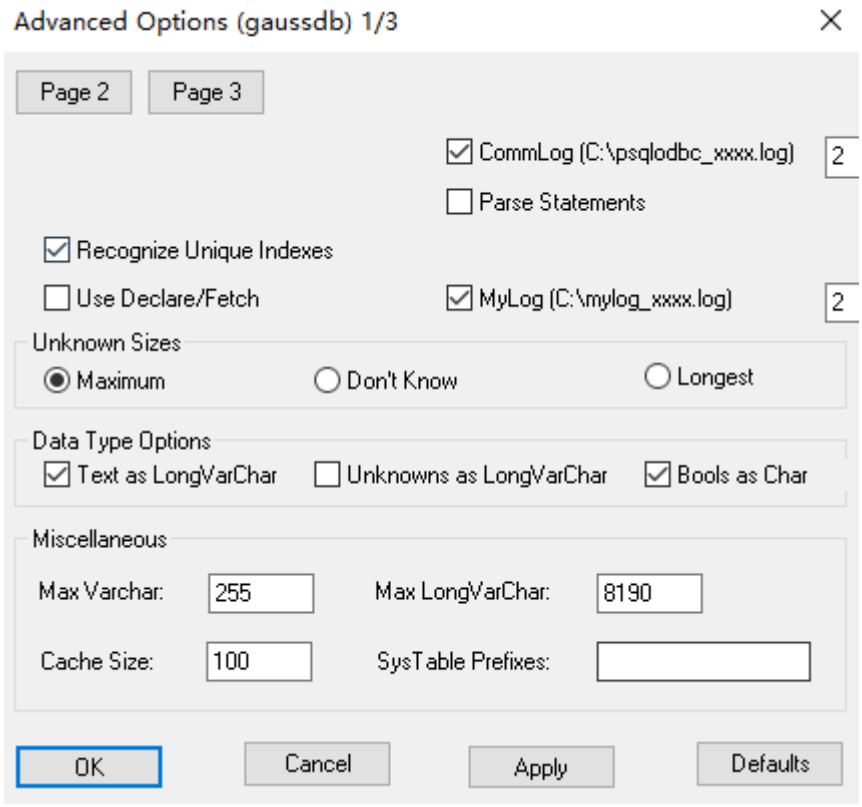
Step 3 Configure a data source.

On the **User DSN** tab page, click **Add** and choose **GaussDB Unicode** for setup.



For details about the parameters, see **Configuring a Data Source in the Linux OS**.

You can click **Datasource** to configure whether to print logs.



NOTICE

The entered username and password will be recorded in the Windows registry and you do not need to enter them again when connecting to the database next time. For security purposes, you are advised to delete sensitive information before clicking **Save** and enter the required username and password again when using ODBC APIs to connect to the database.

Step 4 Enable the SSL mode.

Change the value of **SSL Mode** in **Step 3** to **require**.

Table 5-33 sslmode options

sslmode	Whether SSL Encryption Is Enabled	Description
disable	No	SSL connection is not enabled.
allow	Possible	If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified.

sslmode	Whether SSL Encryption Is Enabled	Description
prefer	Possible	If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified.
require	Yes	SSL connection is required and data is encrypted. However, authenticity of the database server will not be verified.
verify-ca	Yes	SSL connection is required and whether the database has a trusted certificate will be verified. Currently, Windows ODBC does not support the cert authentication.
verify-full	Yes	SSL connection is required. In addition to the check scope specified by **verify-ca**, the system checks whether the name of the host where the database resides is the same as that in the certificate. Currently, Windows ODBC does not support the cert authentication.

Step 5 Configure a GaussDB server. For details, contact the administrator.

Step 6 Restart the database instance.

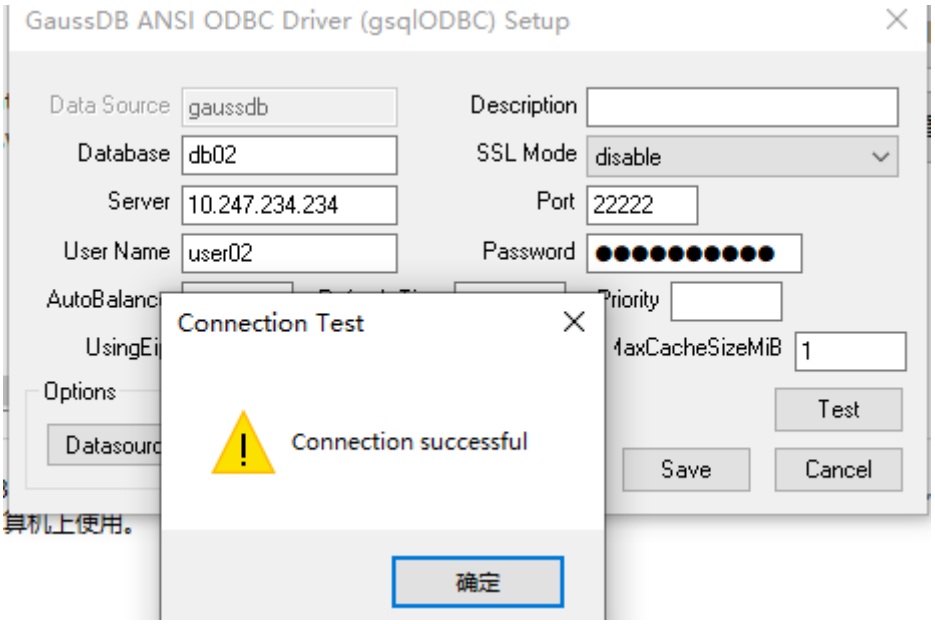
```
gs_om -t stop
gs_om -t start
```

----End

Verifying the Data Source Configuration

Click **Test**.

- If the following information is displayed, the configuration is correct and the connection succeeds.



- If error information is displayed, the configuration is incorrect. Check the configuration.

FAQs

- connect to server failed: no such file or directory

  Possible causes:

  – An incorrect or unreachable database IP address or port was configured.

  Check the **Server** and **Port** configuration items in data sources.

  – Server monitoring is improper.

  If **Server** and **Port** are correctly configured, ensure the proper NIC and port are monitored by following the database server configurations in the procedure in this section.

  – Firewall and network gatekeeper settings are improper.

  Check firewall settings, and ensure that the database communication port is trusted.

  Check to ensure that network gatekeeper settings are proper (if any).

- The password-stored method is not supported.

  Possible causes:

  sslmode is not configured for the data source. Set this configuration item to **allow** or a higher level to enable SSL connections. For details on **sslmode**, see **Table 5-33**.

- authentication method 10 not supported.

  If this error occurs on an open-source client, the cause may be:

  The database stores only the SHA-256 hash of the password, but the open-source client supports only MD5 hashes.

NOTE

- The database stores the hashes of user passwords instead of actual passwords.
- If a password is updated or a user is created, both types of hashes will be stored, compatible with open-source authentication protocols.
- An MD5 hash can only be generated using the original password, but the password cannot be obtained by reversing its SHA-256 hash. Passwords in the source version will only have SHA-256 hashes and not support MD5 authentication.
- The MD5 encryption algorithm has lower security and poses security risks. Therefore, you are advised to use a more secure encryption algorithm.

To solve this problem, you can update the user password (see **ALTER USER**) or create a user (see **CREATE USER**) having the same permissions as the faulty user.

- unsupported frontend protocol 3.51: server supports 1.0 to 3.0

  The database version is too early or the database is an open-source database. Use the driver of the required version to connect to the database.

- FATAL: GSS authentication method is not allowed because XXXX user password is not disabled.

  In **gs_hba.conf** of the target DN, the authentication mode is set to **gss** for authenticating the IP address of the current client. However, this authentication algorithm cannot authenticate clients. Change the authentication algorithm to **sha256** and try again. For details, see **Step 5**.

5.4.5 Example: Common Functions and Batch Binding

NOTE

- The following is an example of the command for compiling ODBC application code in Windows:

  ```
  gcc  odbctest.c -o odbctest -lodbc32
  ```

  Run the following command:

  ```
  ./odbctest.exe
  ```

- The following is an example of the command for compiling ODBC application code in Linux:

  ```
  gcc odbctest.c -o odbctest -lodbc
  ```

  Run the following command:

  ```
  ./odbctest
  ```

- If **sql.h** or API cannot be found during compilation, manually connect to the header file and dynamic library of unixODBC.

  ```
  gcc -I /home/omm/unixodbc/include -L /home/omm/unixodbc/lib odbctest.c -o odbctest -lodbc
  ```

Code for Common Functions

```
// The example shows how to obtain data from GaussDB through the ODBC interface.
// DBtest.c (compile with: libodbc.so)
#ifdef WIN32
#include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
SQLHENV      V_OD_Env;       // Handle ODBC environment
SQLHSTMT     V_OD_hstmt;     // Handle statement
SQLHDBC      V_OD_hdbc;      // Handle connection
char         typename[100];
```



```
SQLINTEGER    value = 100;
SQLINTEGER    V_OD_erg,V_OD_buffer,V_OD_err,V_OD_id;
int main(int argc,char *argv[])
{
    // 1. Allocate an environment handle.
    V_OD_erg = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&V_OD_Env);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
        printf("Error AllocHandle\n");
        exit(0);
    }
    // 2. Set environment attributes (version information).
    SQLSetEnvAttr(V_OD_Env, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);
    // 3. Allocate a connection handle.
    V_OD_erg = SQLAllocHandle(SQL_HANDLE_DBC, V_OD_Env, &V_OD_hdbc);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
        SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
        exit(0);
    }
   // In this example, the username and password are stored in environment variables. Before running this example, set environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in the local environment (set the environment variable names based on the actual situation).
    char *userName;
    userName = getenv("EXAMPLE_USERNAME_ENV");
    char *password;
    password = getenv("EXAMPLE_PASSWORD_ENV");
    // 4. Set connection attributes.
    SQLSetConnectAttr(V_OD_hdbc, SQL_ATTR_AUTOCOMMIT,(SQLPOINTER *)SQL_AUTOCOMMIT_ON, 0);
    // 5. Connect to the data source. userName and password indicate the username and password for connecting to the database, respectively.
   // If the username and password have been set in the odbc.ini file, you can retain "". However, you are advised not to do so because the username and password will be disclosed if the permission for odbc.ini is abused.
    V_OD_erg = SQLConnect(V_OD_hdbc, (SQLCHAR*) "gaussdb", SQL_NTS,
                    (SQLCHAR*) userName, SQL_NTS,  (SQLCHAR*) password, SQL_NTS);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
       printf("Error SQLConnect %d\n",V_OD_erg);
       SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
       exit(0);
    }
    printf("Connected !\n");
    // 6. Set statement attributes.
    SQLSetStmtAttr(V_OD_hstmt,SQL_ATTR_QUERY_TIMEOUT,(SQLPOINTER *)3,0);
     // 7. Allocate a statement handle.
    SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
    // 8. Run SQL statements.
    SQLExecDirect(V_OD_hstmt,"drop table IF EXISTS customer_t1",SQL_NTS);
    SQLExecDirect(V_OD_hstmt,"CREATE TABLE customer_t1(c_customer_sk INTEGER, c_customer_name VARCHAR(32));",SQL_NTS);
    SQLExecDirect(V_OD_hstmt,"insert into customer_t1 values(25,'li')",SQL_NTS);
    // 9. Prepare for execution.
    SQLPrepare(V_OD_hstmt,"insert into customer_t1 values(?)",SQL_NTS);
    // 10. Bind parameters.
    SQLBindParameter(V_OD_hstmt,1,SQL_PARAM_INPUT,SQL_C_SLONG,SQL_INTEGER,0,0,
                  &value,0,NULL);
    // 11. Run prepared statements.
    SQLExecute(V_OD_hstmt);
    SQLExecDirect(V_OD_hstmt,"select c_customer_sk from customer_t1",SQL_NTS);
    // 12. Obtain attributes of a specific column in the result set.
    SQLColAttribute(V_OD_hstmt,1,SQL_DESC_TYPE,typename,100,NULL,NULL);
    printf("SQLColAtrribute %s\n",typename);
    // 13. Bind the result set.
    SQLBindCol(V_OD_hstmt,1,SQL_C_SLONG, (SQLPOINTER)&V_OD_buffer,150,
            (SQLLEN *)&V_OD_err);
    // 14. Obtain data in the result set by executing SQLFetch.
    V_OD_erg=SQLFetch(V_OD_hstmt);
```



```
    // 15. Obtain and return data by executing SQLGetData.
    while(V_OD_erg != SQL_NO_DATA)
    {
        SQLGetData(V_OD_hstmt,1,SQL_C_SLONG,(SQLPOINTER)&V_OD_id,0,NULL);
        printf("SQLGetData ----ID = %d\n",V_OD_id);
        V_OD_erg=SQLFetch(V_OD_hstmt);
    };
    printf("Done !\n");
    // 16. Disconnect data source connections and release handles.
    SQLFreeHandle(SQL_HANDLE_STMT,V_OD_hstmt);
    SQLDisconnect(V_OD_hdbc);
    SQLFreeHandle(SQL_HANDLE_DBC,V_OD_hdbc);
    SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
    return(0);
}
```

Code for Batch Processing

```
/*************************************************************************
* Enable UseBatchProtocol in the data source and set the database parameter support_batch_bind to on.
* The CHECK_ERROR command is used to check and print error information.
* This example is used to interactively obtain the DSN, data volume to be processed, and volume of ignored data from users, and insert required data into the test_odbc_batch_insert table.
*************************************************************************/
#ifdef WIN32
#include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
#include <string.h>
void Exec(SQLHDBC hdbc, SQLCHAR* sql)
{
    SQLRETURN retcode;                  // Return status
    SQLHSTMT hstmt = SQL_NULL_HSTMT;    // Statement handle
    SQLCHAR     loginfo[2048];
    // Allocate Statement Handle
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLAllocHandle(SQL_HANDLE_STMT) failed");
        return;
    }
    // Prepare Statement
    retcode = SQLPrepare(hstmt, (SQLCHAR*) sql, SQL_NTS);
    sprintf((char*)loginfo, "SQLPrepare log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLPrepare(hstmt, (SQLCHAR*) sql, SQL_NTS) failed");
        return;
    }
    // Execute Statement
    retcode = SQLExecute(hstmt);
    sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLExecute(hstmt) failed");
        return;
    }
    // Free Handle
    retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
    sprintf((char*)loginfo, "SQLFreeHandle stmt log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLFreeHandle(SQL_HANDLE_STMT, hstmt) failed");
        return;
    }
}
```



```
int main ()
{
    SQLHENV  henv  = SQL_NULL_HENV;
    SQLHDBC  hdbc  = SQL_NULL_HDBC;
    long     int      batchCount = 1000; // Amount of data that is bound in batches
    SQLLEN   rowsCount = 0;
    int      ignoreCount = 0; // Amount of data that is not imported to the database among the data that is
bound in batches
    int      i = 0;
    SQLRETURN   retcode;
    SQLCHAR     dsn[1024] = {'\0'};
    SQLCHAR     loginfo[2048];
    do
    {
        if (ignoreCount > batchCount)
        {
            printf("ignoreCount(%d) should be less than batchCount(%d)\n", ignoreCount, batchCount);
        }
    }while(ignoreCount > batchCount);
    retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLAllocHandle failed");
        goto exit;
    }
    // Set ODBC version.
    retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                                    (SQLPOINTER*)SQL_OV_ODBC3, 0);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetEnvAttr failed");
        goto exit;
    }
    // Allocate Connection
    retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLAllocHandle failed");
        goto exit;
    }
    // Set Login Timeout
    retcode = SQLSetConnectAttr(hdbc, SQL_LOGIN_TIMEOUT, (SQLPOINTER)5, 0);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetConnectAttr failed");
        goto exit;
    }
    // Set Auto Commit
    retcode = SQLSetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT,
                                    (SQLPOINTER)(1), 0);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetConnectAttr failed");
        goto exit;
    }
    // Connect to DSN
    // Replace GaussDB with the name of the data source used by the user.
    sprintf(loginfo, "SQLConnect(DSN:%s)", dsn);
    retcode = SQLConnect(hdbc, (SQLCHAR*) "gaussdb", SQL_NTS,
                               (SQLCHAR*) NULL, 0, NULL, 0);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLConnect failed");
        goto exit;
    }
    // init table info.
    Exec(hdbc, "drop table if exists test_odbc_batch_insert");
    Exec(hdbc, "create table test_odbc_batch_insert(id int primary key, col varchar2(50))");
   // The code constructs the data to be inserted based on the data volume entered by users.
```



```
  {
      SQLRETURN retcode;
      SQLHSTMT hstmtinesrt = SQL_NULL_HSTMT;
      SQLCHAR      *sql = NULL;
      SQLINTEGER   *ids  = NULL;
      SQLCHAR      *cols = NULL;
      SQLLEN       *bufLenIds = NULL;
      SQLLEN       *bufLenCols = NULL;
      SQLUSMALLINT *operptr = NULL;
      SQLUSMALLINT *statusptr = NULL;
      SQLULEN      process = 0;
      // Data is constructed by column. Each column is stored continuously.
      ids = (SQLINTEGER*)malloc(sizeof(ids[0]) * batchCount);
      cols = (SQLCHAR*)malloc(sizeof(cols[0]) * batchCount * 50);
      // Data size in each row for a column
      bufLenIds = (SQLLEN*)malloc(sizeof(bufLenIds[0]) * batchCount);
      bufLenCols = (SQLLEN*)malloc(sizeof(bufLenCols[0]) * batchCount);
      // The value SQL_PARAM_IGNORE or SQL_PARAM_PROCEED specifies whether a row needs to be
processed.
      operptr = (SQLUSMALLINT*)malloc(sizeof(operptr[0]) * batchCount);
      memset(operptr, 0, sizeof(operptr[0]) * batchCount);
      // Processing result of the row
      // Note: In the database, a statement belongs to one transaction. Therefore, data is processed as a
unit. Either all data is inserted successfully or all data fails to be inserted.
      statusptr = (SQLUSMALLINT*)malloc(sizeof(statusptr[0]) * batchCount);
      memset(statusptr, 88, sizeof(statusptr[0]) * batchCount);
      if (NULL == ids || NULL == cols || NULL == bufLenCols || NULL == bufLenIds)
      {
          fprintf(stderr, "FAILED:\tmalloc data memory failed\n");
          goto exit;
      }
      for (i = 0; i < batchCount; i++)
      {
          ids[i] = i;
          sprintf(cols + 50 * i, "column test value %d", i);
          bufLenIds[i] = sizeof(ids[i]);
          bufLenCols[i] = strlen(cols + 50 * i);
          operptr[i] = (i < ignoreCount) ? SQL_PARAM_IGNORE : SQL_PARAM_PROCEED;
      }
      // Allocate Statement Handle
      retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmtinesrt);

      if (!SQL_SUCCEEDED(retcode)) {
          printf("SQLAllocHandle failed");
          goto exit;
      }
      // Prepare Statement
      sql = (SQLCHAR*)"insert into test_odbc_batch_insert values(?, ?)";
      retcode = SQLPrepare(hstmtinesrt, (SQLCHAR*) sql, SQL_NTS);
      sprintf((char*)loginfo, "SQLPrepare log: %s", (char*)sql);

      if (!SQL_SUCCEEDED(retcode)) {
          printf("SQLPrepare failed");
          goto exit;
      }
      retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER)batchCount,
sizeof(batchCount));

      if (!SQL_SUCCEEDED(retcode)) {
          printf("SQLSetStmtAttr failed");
          goto exit;
      }
      retcode = SQLBindParameter(hstmtinesrt, 1, SQL_PARAM_INPUT, SQL_C_SLONG, SQL_INTEGER,
sizeof(ids[0]), 0,&(ids[0]), 0, bufLenIds);

      if (!SQL_SUCCEEDED(retcode)) {
          printf("SQLBindParameter failed");
          goto exit;
      }
```



```
    retcode = SQLBindParameter(hstmtinesrt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, 50, 50,
cols, 50, bufLenCols);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLBindParameter failed");
        goto exit;
    }
    retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAMS_PROCESSED_PTR, (SQLPOINTER)&process,
sizeof(process));

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetStmtAttr failed");
        goto exit;
    }
    retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_STATUS_PTR, (SQLPOINTER)statusptr,
sizeof(statusptr[0]) * batchCount);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetStmtAttr failed");
        goto exit;
    }
    retcode = SQLSetStmtAttr(hstmtinesrt, SQL_ATTR_PARAM_OPERATION_PTR, (SQLPOINTER)operptr,
sizeof(operptr[0]) * batchCount);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLSetStmtAttr failed");
        goto exit;
    }
    retcode = SQLExecute(hstmtinesrt);
    sprintf((char*)loginfo, "SQLExecute stmt log: %s", (char*)sql);

    if (!SQL_SUCCEEDED(retcode)) {
        printf("SQLExecute(hstmtinesrt) failed");
        goto exit;
        retcode = SQLRowCount(hstmtinesrt, &rowsCount);

        if (!SQL_SUCCEEDED(retcode)) {
            printf("SQLRowCount failed");
            goto exit;
        }
        if (rowsCount != (batchCount - ignoreCount))
        {
            sprintf(loginfo, "(batchCount - ignoreCount)(%d) != rowsCount(%d)", (batchCount -
ignoreCount), rowsCount);

            if (!SQL_SUCCEEDED(retcode)) {
                printf("SQLExecute failed");
                goto exit;
            }
        }
        else
        {
            sprintf(loginfo, "(batchCount - ignoreCount)(%d) == rowsCount(%d)", (batchCount -
ignoreCount), rowsCount);

            if (!SQL_SUCCEEDED(retcode)) {
                printf("SQLExecute failed");
                goto exit;
            }
        }
        // check row number returned
        if (rowsCount != process)
        {
            sprintf(loginfo, "process(%d) != rowsCount(%d)", process, rowsCount);

            if (!SQL_SUCCEEDED(retcode)) {
                printf("SQLExecute failed");
                goto exit;
            }
```



```
        }
        else
        {
            sprintf(loginfo, "process(%d) == rowsCount(%d)", process, rowsCount);

            if (!SQL_SUCCEEDED(retcode)) {
                printf("SQLExecute failed");
                goto exit;
            }
        }
        for (i = 0; i < batchCount; i++)
        {
            if (i < ignoreCount)
            {
                if (statusptr[i] != SQL_PARAM_UNUSED)
                {
                    sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_UNUSED", i, statusptr[i]);

                    if (!SQL_SUCCEEDED(retcode)) {
                        printf("SQLExecute failed");
                        goto exit;
                    }
                }
            }
            else if (statusptr[i] != SQL_PARAM_SUCCESS)
            {
                sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_SUCCESS", i, statusptr[i]);

                if (!SQL_SUCCEEDED(retcode)) {
                    printf("SQLExecute failed");
                    goto exit;
                }
            }
        }
        retcode = SQLFreeHandle(SQL_HANDLE_STMT, hstmtinesrt);
        sprintf((char*)loginfo, "SQLFreeHandle hstmtinesrt");

        if (!SQL_SUCCEEDED(retcode)) {
            printf("SQLFreeHandle failed");
            goto exit;
        }
    }
  }
exit:
  (void) printf ("\nComplete.\n");
  // Connection
  if (hdbc != SQL_NULL_HDBC) {
      SQLDisconnect(hdbc);
      SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
  }
  // Environment
  if (henv != SQL_NULL_HENV)
      SQLFreeHandle(SQL_HANDLE_ENV, henv);
  return 0;
}
```

5.4.6 Typical Application Scenarios and Configurations

Log Diagnosis Scenario

ODBC logs are classified into unixODBC driver manager logs and gsqlODBC driver logs. The former is used to trace whether the application API is successfully executed, and the latter is used to locate problems based on DFX logs generated during underlying implementation.

The unixODBC log needs to be configured in the **odbcinst.ini** file:

```
[ODBC]
Trace=Yes
TraceFile=/path/to/odbctrace.log

[GaussMPP]
Driver64=/usr/local/lib/gsqlodbcw.so
setup=/usr/local/lib/gsqlodbcw.so
```

For gsqlODBC logs, you only need to add the following content to **odbc.ini**:

```
[gaussdb]
Driver=GaussMPP
Servername=10.10.0.13 # Database server IP address
...
Debug=1 # Enable the debug log function of the driver.
```

NOTE

The unixODBC logs are generated in the path configured by **TraceFile**. The gsqlODBC generates the **mylog_*xxx*.log** file in the **/tmp/** directory.

High Performance

If a large amount of data needs to be inserted, you are advised to perform the following operations:

- You need to set **UseBatchProtocol** to **1** in the **odbc.ini** file and **support_batch_bind** to **on** in the database.
- The ODBC program binding type must be the same as that in the database.
- The character set of the client is the same as that of the database.
- The transaction is committed manually.

odbc.ini configuration file:

```
[gaussdb]
Driver=GaussMPP
Servername=10.10.0.13 # Database server IP address
...
UseBatchProtocol=1 # Enabled by default
ConnSettings=set client_encoding=UTF8 # Set the character code on the client to be the same as that on the server.
```

Binding type case:

```
#ifdef WIN32
#include <windows.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
#include <string.h>
#include <sys/time.h>

#define MESSAGE_BUFFER_LEN  128
SQLHANDLE   h_env = NULL;
SQLHANDLE   h_conn = NULL;
SQLHANDLE   h_stmt = NULL;
void print_error()
{
    SQLCHAR     Sqlstate[SQL_SQLSTATE_SIZE+1];
    SQLINTEGER  NativeError;
    SQLCHAR     MessageText[MESSAGE_BUFFER_LEN];
    SQLSMALLINT TextLength;
    SQLRETURN   ret = SQL_ERROR;
```



```
    ret = SQLGetDiagRec(SQL_HANDLE_STMT, h_stmt, 1, Sqlstate, &NativeError, MessageText,
MESSAGE_BUFFER_LEN, &TextLength);
    if ( SQL_SUCCESS == ret)
    {
        printf("\n STMT ERROR-%05d %s", NativeError, MessageText);
        return;
    }

    ret = SQLGetDiagRec(SQL_HANDLE_DBC, h_conn, 1, Sqlstate, &NativeError, MessageText,
MESSAGE_BUFFER_LEN, &TextLength);
    if ( SQL_SUCCESS == ret)
    {
        printf("\n CONN ERROR-%05d %s", NativeError, MessageText);
        return;
    }

    ret = SQLGetDiagRec(SQL_HANDLE_ENV, h_env, 1, Sqlstate, &NativeError, MessageText,
MESSAGE_BUFFER_LEN, &TextLength);
    if ( SQL_SUCCESS == ret)
    {
        printf("\n ENV ERROR-%05d %s", NativeError, MessageText);
        return;
    }

    return;
}

/* Expect the function to return SQL_SUCCESS. */
#define RETURN_IF_NOT_SUCCESS(func) \
{\
    SQLRETURN ret_value = (func);\
    if (SQL_SUCCESS != ret_value)\
    {\
        print_error();\
        printf("\n failed line = %u: expect SQL_SUCCESS, but ret = %d", __LINE__, ret_value);\
        return SQL_ERROR; \
    }\
}

/* Expect the function to return SQL_SUCCESS. */
#define RETURN_IF_NOT_SUCCESS_I(i, func) \
{\
    SQLRETURN ret_value = (func);\
    if (SQL_SUCCESS != ret_value)\
    {\
        print_error();\
        printf("\n failed line = %u (i=%d): : expect SQL_SUCCESS, but ret = %d", __LINE__, (i), ret_value);\
        return SQL_ERROR; \
    }\
}

/* Expect the function to return SQL_SUCCESS_WITH_INFO. */
#define RETURN_IF_NOT_SUCCESS_INFO(func) \
{\
    SQLRETURN ret_value = (func);\
    if (SQL_SUCCESS_WITH_INFO != ret_value)\
    {\
        print_error();\
        printf("\n failed line = %u: expect SQL_SUCCESS_WITH_INFO, but ret = %d", __LINE__, ret_value);\
        return SQL_ERROR; \
    }\
}

/* Expect the values are the same. */
#define RETURN_IF_NOT(expect, value) \
if ((expect) != (value))\
{\
    printf("\n failed line = %u: expect = %u, but value = %u", __LINE__, (expect), (value)); \
    return SQL_ERROR;\
```



```
}

/* Expect the character strings are the same. */
#define RETURN_IF_NOT_STRCMP_I(i, expect, value) \
if (( NULL == (expect) ) || (NULL == (value)))\
{\
    printf("\n failed line = %u (i=%u): input NULL pointer !", __LINE__, (i)); \
    return SQL_ERROR; \
}\
else if (0 != strcmp((expect), (value)))\
{\
    printf("\n failed line = %u (i=%u): expect = %s, but value = %s", __LINE__, (i), (expect), (value)); \
    return SQL_ERROR;\
}


// prepare + execute SQL statement
int execute_cmd(SQLCHAR *sql)
{
    if ( NULL == sql )
    {
        return SQL_ERROR;
    }

    if ( SQL_SUCCESS != SQLPrepare(h_stmt, sql, SQL_NTS))
    {
        return SQL_ERROR;
    }

    if ( SQL_SUCCESS != SQLExecute(h_stmt))
    {
        return SQL_ERROR;
    }

    return SQL_SUCCESS;
}
// execute + commit handle
int commit_exec()
{
    if ( SQL_SUCCESS != SQLExecute(h_stmt))
    {
        return SQL_ERROR;
    }

     // Manual commit
    if ( SQL_SUCCESS != SQLEndTran(SQL_HANDLE_DBC, h_conn, SQL_COMMIT))
    {
        return SQL_ERROR;
    }

    return SQL_SUCCESS;
}

int begin_unit_test()
{
    SQLINTEGER    ret;

    /* Allocate an environment handle. */
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &h_env);
    if ((SQL_SUCCESS != ret) && (SQL_SUCCESS_WITH_INFO != ret))
    {
        printf("\n begin_unit_test::SQLAllocHandle SQL_HANDLE_ENV failed ! ret = %d", ret);
        return SQL_ERROR;
    }

    /* Set the version number before connection. */
    if (SQL_SUCCESS != SQLSetEnvAttr(h_env, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)SQL_OV_ODBC3,
0))
    {
```



```
        print_error();
        printf("\n begin_unit_test::SQLSetEnvAttr SQL_ATTR_ODBC_VERSION failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    /* Allocate a connection handle. */
    ret = SQLAllocHandle(SQL_HANDLE_DBC, h_env, &h_conn);
    if (SQL_SUCCESS != ret)
    {
        print_error();
        printf("\n begin_unit_test::SQLAllocHandle SQL_HANDLE_DBC failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    /* Establish a connection. */
 ret = SQLConnect(h_conn, (SQLCHAR*) "gaussdb", SQL_NTS,
                          (SQLCHAR*) NULL, 0, NULL, 0);
    if (SQL_SUCCESS != ret)
    {
        print_error();
        printf("\n begin_unit_test::SQLConnect  failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_DBC, h_conn);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    /* Allocate a statement handle. */
    ret = SQLAllocHandle(SQL_HANDLE_STMT, h_conn, &h_stmt);
    if (SQL_SUCCESS != ret)
    {
        print_error();
        printf("\n begin_unit_test::SQLAllocHandle SQL_HANDLE_STMT failed ! ret = %d", ret);
        SQLFreeHandle(SQL_HANDLE_DBC, h_conn);
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
        return SQL_ERROR;
    }

    return SQL_SUCCESS;
}

void end_unit_test()
{
    /* Release a statement handle. */
    if (NULL != h_stmt)
    {
        SQLFreeHandle(SQL_HANDLE_STMT, h_stmt);
    }

    /* Release a connection handle. */
    if (NULL != h_conn)
    {
        SQLDisconnect(h_conn);
        SQLFreeHandle(SQL_HANDLE_DBC, h_conn);
    }

    /* Release an environment handle. */
    if (NULL != h_env)
    {
        SQLFreeHandle(SQL_HANDLE_ENV, h_env);
    }

    return;
}

int main()
{
 // begin test
```



```
if (begin_unit_test() != SQL_SUCCESS)
  {
     printf("\n begin_test_unit failed.");
     return SQL_ERROR;
  }
  // The handle configuration is the same as that in the preceding case
  int         i = 0;
  SQLCHAR*    sql_drop   = "drop table if exists test_bindnumber_001";
  SQLCHAR*    sql_create = "create table test_bindnumber_001("
                           "f4 number, f5 number(10, 2)"
                           ")";
  SQLCHAR*    sql_insert = "insert into test_bindnumber_001 values(?, ?)";
  SQLCHAR*    sql_select = "select * from test_bindnumber_001";
  SQLLEN      RowCount;
  SQL_NUMERIC_STRUCT st_number;
  SQLCHAR     getValue[2][MESSAGE_BUFFER_LEN];

  /* Step 1. Create a table. */
  RETURN_IF_NOT_SUCCESS(execute_cmd(sql_drop));
  RETURN_IF_NOT_SUCCESS(execute_cmd(sql_create));

  /* Step 2.1 Bind parameters using the SQL_NUMERIC_STRUCT structure. */
  RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));

  // First line: 1234.5678
  memset(st_number.val, 0, SQL_MAX_NUMERIC_LEN);
  st_number.precision = 8;
  st_number.scale = 4;
  st_number.sign = 1;
  st_number.val[0] = 0x4E;
  st_number.val[1] = 0x61;
  st_number.val[2] = 0xBC;

  RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));
  RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));

  // Disable the automatic commit function.
  SQLSetConnectAttr(h_conn, SQL_ATTR_AUTOCOMMIT, (SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0);

  RETURN_IF_NOT_SUCCESS(commit_exec());
  RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
  RETURN_IF_NOT(1, RowCount);

  // Second line: 12345678
  memset(st_number.val, 0, SQL_MAX_NUMERIC_LEN);
  st_number.precision = 8;
  st_number.scale = 0;
  st_number.sign = 1;
  st_number.val[0] = 0x4E;
  st_number.val[1] = 0x61;
  st_number.val[2] = 0xBC;

  RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 0, &st_number, 0, NULL));
  RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 0, &st_number, 0, NULL));
  RETURN_IF_NOT_SUCCESS(commit_exec());
  RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
  RETURN_IF_NOT(1, RowCount);

  // Third line: 12345678
  memset(st_number.val, 0, SQL_MAX_NUMERIC_LEN);
  st_number.precision = 0;
  st_number.scale = 4;
  st_number.sign = 1;
  st_number.val[0] = 0x4E;
  st_number.val[1] = 0x61;
```



```
    st_number.val[2] = 0xBC;

    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_NUMERIC,
SQL_NUMERIC, sizeof(SQL_NUMERIC_STRUCT), 4, &st_number, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);


    /* Step 2.2 Bind parameters by using the SQL_C_CHAR character string in the fourth line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    SQLCHAR*    szNumber = "1234.5678";
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_NUMERIC, strlen(szNumber), 0, szNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_NUMERIC, strlen(szNumber), 0, szNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Step 2.3 Bind parameters by using SQL_C_FLOAT in the fifth line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    SQLREAL    fNumber = 1234.5678;
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_FLOAT,
SQL_NUMERIC, sizeof(fNumber), 4, &fNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_FLOAT,
SQL_NUMERIC, sizeof(fNumber), 4, &fNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Step 2.4 Bind parameters by using SQL_C_DOUBLE in the sixth line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    SQLDOUBLE   dNumber = 1234.5678;
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_DOUBLE,
SQL_NUMERIC, sizeof(dNumber), 4, &dNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_DOUBLE,
SQL_NUMERIC, sizeof(dNumber), 4, &dNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    SQLBIGINT   bNumber1 = 0xFFFFFFFFFFFFFFFF;
    SQLBIGINT   bNumber2 = 12345;

    /* Step 2.5 Bind parameters by using SQL_C_SBIGINT in the seventh line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_SBIGINT,
SQL_NUMERIC, sizeof(bNumber1), 4, &bNumber1, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_SBIGINT,
SQL_NUMERIC, sizeof(bNumber2), 4, &bNumber2, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Step 2.6 Bind parameters by using SQL_C_UBIGINT in the eighth line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_UBIGINT,
SQL_NUMERIC, sizeof(bNumber1), 4, &bNumber1, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_UBIGINT,
SQL_NUMERIC, sizeof(bNumber2), 4, &bNumber2, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    SQLLEN  lNumber1 = 0xFFFFFFFFFFFFFFFF;
    SQLLEN  lNumber2 = 12345;
```



```
    /* Step 2.7 Bind parameters by using SQL_C_LONG in the ninth line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_NUMERIC, sizeof(lNumber1), 0, &lNumber1, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_NUMERIC, sizeof(lNumber2), 0, &lNumber2, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Step 2.8 Bind parameters by using SQL_C_ULONG in the tenth line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_ULONG,
SQL_NUMERIC, sizeof(lNumber1), 0, &lNumber1, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_ULONG,
SQL_NUMERIC, sizeof(lNumber2), 0, &lNumber2, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    SQLSMALLINT sNumber = 0xFFFF;

    /* Step 2.9 Bind parameters by using SQL_C_SHORT in the eleventh line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_SHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_SHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Step 2.10 Bind parameters by using SQL_C_USHORT in the twelfth line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_USHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_USHORT,
SQL_NUMERIC, sizeof(sNumber), 0, &sNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    SQLCHAR cNumber = 0xFF;

    /* Step 2.11 Bind parameters by using SQL_C_TINYINT in the thirteenth line. */
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_TINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_TINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Step 2.12 Bind parameters by using SQL_C_UTINYINT in the fourteenth line.*/
    RETURN_IF_NOT_SUCCESS(SQLPrepare(h_stmt, sql_insert, SQL_NTS));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 1, SQL_PARAM_INPUT, SQL_C_UTINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(SQLBindParameter(h_stmt, 2, SQL_PARAM_INPUT, SQL_C_UTINYINT,
SQL_NUMERIC, sizeof(cNumber), 0, &cNumber, 0, NULL));
    RETURN_IF_NOT_SUCCESS(commit_exec());
    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(1, RowCount);

    /* Use the character string type to unify the expectation. */
    SQLCHAR*    expectValue[14][2] = {{"1234.5678",            "1234.57"},
                                      {"12345678",             "12345678"},
                                      {"0",                    "0"},
```

```
                                {"1234.5678",           "1234.57"},
                                {"1234.5677",           "1234.57"},
                                {"1234.5678",           "1234.57"},
                                {"-1",                  "12345"},
                                {"18446744073709551615",  "12345"},
                                {"-1",                  "12345"},
                                {"4294967295",            "12345"},
                                {"-1",                  "-1"},
                                {"65535",                 "65535"},
                                {"-1",                  "-1"},
                                {"255",                  "255"},
                                };

    RETURN_IF_NOT_SUCCESS(execute_cmd(sql_select));
    while ( SQL_NO_DATA != SQLFetch(h_stmt))
    {
        RETURN_IF_NOT_SUCCESS_I(i, SQLGetData(h_stmt, 1, SQL_C_CHAR, &getValue[0],
MESSAGE_BUFFER_LEN, NULL));
        RETURN_IF_NOT_SUCCESS_I(i, SQLGetData(h_stmt, 2, SQL_C_CHAR, &getValue[1],
MESSAGE_BUFFER_LEN, NULL));

        //RETURN_IF_NOT_STRCMP_I(i, expectValue[i][0], getValue[0]);
        //RETURN_IF_NOT_STRCMP_I(i, expectValue[i][1], getValue[1]);
        i++;
    }

    RETURN_IF_NOT_SUCCESS(SQLRowCount(h_stmt, &RowCount));
    RETURN_IF_NOT(i, RowCount);
 SQLCloseCursor(h_stmt);
    /* Final step. Delete the table and restore the environment. */
    RETURN_IF_NOT_SUCCESS(execute_cmd(sql_drop));

    end_unit_test();
}
```

NOTE

In the preceding example, the **number** column is defined. When the SQLBindParameter API is called, the performance of binding SQL_NUMERIC is higher than that of SQL_LONG. If char is used, the data type needs to be converted when data is inserted to the database server, causing a performance bottleneck.

Automatic Primary/Standby Switchover

Example Scenario

If a database instance is configured with one primary DN and multiple standby DNs, write the IP addresses of all DNs into the configuration file. ODBC automatically searches for the primary DN and establishes a connection with it. When a primary/standby switchover occurs, ODBC can also connect to the new primary DN.

5.4.7 ODBC Interface Reference

The ODBC interface is a set of API functions provided to users. This chapter describes its common interfaces. For details on other interfaces, see "ODBC Programmer's Reference" at MSDN (https://msdn.microsoft.com/en-us/library/windows/desktop/ms714177(v=vs.85).aspx).

5.4.7.1 SQLAllocEnv

In ODBC 3.x, SQLAllocEnv (an ODBC 2.x function) was deprecated and replaced by SQLAllocHandle. For details, see **SQLAllocHandle**.

5.4.7.2 SQLAllocConnect

In ODBC 3.x, SQLAllocConnect (an ODBC 2.x function) was deprecated and replaced by SQLAllocHandle. For details, see **SQLAllocHandle**.

5.4.7.3 SQLAllocHandle

Description

Allocates environment, connection, statement, or descriptor handles. This function replaces the deprecated ODBC 2.x functions SQLAllocEnv, SQLAllocConnect, and SQLAllocStmt.

Prototype

```
SQLRETURN SQLAllocHandle(SQLSMALLINT   HandleType,
                         SQLHANDLE     InputHandle,
                         SQLHANDLE     *OutputHandlePtr);
```

Parameters

Table 5-34 SQLAllocHandle parameters

Keyword	Description
HandleType	Type of handle to be allocated by SQLAllocHandle. The value must be one of the following: • SQL_HANDLE_ENV (environment handle) • SQL_HANDLE_DBC (connection handle) • SQL_HANDLE_STMT (statement handle) • **SQL_HANDLE_DESC** (descriptor handle) The handle allocation sequence is: environment handle > connection handle > statement handle. A later allocated handle depends on a previously allocated handle.
InputHandle	Existing handle to use as a context for the new handle being allocated. The **InputHandle** parameter specifies the parent handle of the handle to be created to establish the hierarchical relationship between handles. Handles of different types have different hierarchical relationships. The **InputHandle** parameter is used to specify the relationship. • If **HandleType** is **SQL_HANDLE_ENV**, the value is **SQL_NULL_HANDLE**, indicating that there is no parent handle. • If **HandleType** is **SQL_HANDLE_DBC**, this must be an environment handle, indicating that the connection handle is created in this environment. • If **HandleType** is set to **SQL_HANDLE_STMT** or **SQL_HANDLE_DESC**, this parameter value must be a connection handle.

Keyword	Description
OutputHandlePtr	**Output parameter**: **OutputHandlePtr** is a pointer that points to the SQLHANDLE type and is used to return a new handle allocated.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.

Precautions

If SQLAllocHandle returns **SQL_ERROR** when it is used to allocate a non-environment handle, it sets **OutputHandlePtr** to **SQL_NULL_HDBC**, **SQL_NULL_HSTMT**, or **SQL_NULL_HDESC**. The application can then call **SQLGetDiagRec**, with **HandleType** and **Handle** set to the value of **IntputHandle**, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

5.4.7.4 SQLAllocStmt

In ODBC 3.x, SQLAllocStmt was deprecated and replaced by SQLAllocHandle. For details, see **SQLAllocHandle**.

5.4.7.5 SQLBindCol

Description

Binds application data buffers to columns in a result set.

Prototype

```
SQLRETURN SQLBindCol(SQLHSTMT       StatementHandle,
                     SQLUSMALLINT   ColumnNumber,
                     SQLSMALLINT    TargetType,
                     SQLPOINTER     TargetValuePtr,
                     SQLLEN     BufferLength,
                     SQLLEN     *StrLen_or_IndPtr);
```


Parameters

Table 5-35 SQLBindCol parameters

Keyword	Description
StatementHandle	Statement handle.
ColumnNumber	Number of the column to be bound. The column number starts with 0 and increases in ascending order. Column 0 is the bookmark column. If no bookmark column is set, column numbers start with 1.
TargetType	C data type in the buffer.
TargetValuePtr	**Output parameter**: pointer to the buffer bound with the column. The SQLFetch function returns data in the buffer. If this parameter is a null pointer, **StrLen_or_IndPtr** is a valid value.
BufferLength	Length of the **TargetValuePtr** buffer in bytes.
StrLen_or_IndPtr	**Output parameter**: pointer to the length or indicator of the buffer. If the value is null, no length or indicator is used.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.

Precautions

If SQLBindCol returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

5.4.7.6 SQLBindParameter

Description

Binds parameter markers in an SQL statement to a buffer.

Prototype

```
SQLRETURN SQLBindParameter(SQLHSTMT       StatementHandle,
                           SQLUSMALLINT   ParameterNumber,
                           SQLSMALLINT    InputOutputType,
                           SQLSMALLINT    ValuetType,
                           SQLSMALLINT    ParameterType,
                           SQLULEN    ColumnSize,
                           SQLSMALLINT    DecimalDigits,
                           SQLPOINTER     ParameterValuePtr,
                           SQLLEN     BufferLength,
                           SQLLEN     *StrLen_or_IndPtr);
```

Parameters

Table 5-36 SQLBindParameter

Keyword	Description
StatementHandle	Statement handle.
ParameterNumber	Parameter marker number, starting with 1 and increasing in ascending order.
InputOutputType	Input/output type of the parameter. The value can be **SQL_PARAM_INPUT**, **SQL_PARAM_OUTPUT**, or **SQL_PARAM_INPUT_OUTPUT**.
ValueType	C data type of the parameter. The value can be **SQL_C_CHAR**, **SQL_C_LONG**, or **SQL_C_DOUBLE**.
ParameterType	SQL data type of the parameter. The value can be **SQL_CHAR**, **SQL_INTEGER**, or **SQL_DOUBLE**.
ColumnSize	Size of the column or expression of the corresponding parameter marker.
DecimalDigits	Decimal digit of the column or the expression of the corresponding parameter marker.
ParameterValuePtr	Pointer to the storage parameter buffer.
BufferLength	Length of the **ParameterValuePtr** buffer in bytes.
StrLen_or_IndPtr	Pointer to the length or indicator of the buffer. If the value is null, no length or indicator is used.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.

Precautions

If SQLBindParameter returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

5.4.7.7 SQLColAttribute

Description

Returns the descriptor information about a column in the result set.

Prototype

```
SQLRETURN SQLColAttibute(SQLHSTMT        StatementHandle,
                         SQLUSMALLINT    ColumnNumber,
                         SQLUSMALLINT    FieldIdentifier,
                         SQLPOINTER      CharacterAtrributePtr,
                         SQLSMALLINT     BufferLength,
                         SQLSMALLINT     *StringLengthPtr,
                         SQLLEN          *NumericAttributePtr);
```

Parameters

Table 5-37 SQLColAttribute parameters

Keyword	Description
StatementHandle	Statement handle.
ColumnNumber	Column number of the field to be queried, starting with 1 and increasing in ascending order.
FieldIdentifier	Field identifier of **ColumnNumber** in IRD.
CharacterAttributePtr	**Output parameter**: pointer to the buffer that returns the **FieldIdentifier** value.

Keyword	Description
BufferLength	• Indicates the length of the buffer if **FieldIdentifier** is an ODBC-defined field and **CharacterAttributePtr** points to a string or a binary buffer. • Ignore this parameter if **FieldIdentifier** is an ODBC-defined field and **CharacterAttributePtr** points to an integer.
StringLengthPtr	**Output parameter**: pointer to a buffer in which the total number of valid bytes (for string data) is stored in ***CharacterAttributePtr**. Ignore the value of **BufferLength** if the data is not a string.
NumericAttributePtr	**Output parameter**: pointer to an integer buffer in which the value of **FieldIdentifier** in the **ColumnNumber** row of the IRD is returned.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.

Precautions

If SQLColAttribute returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

5.4.7.8 SQLConnect

Description

Establishes a connection between a driver and a data source. After the connection is established, the connection handle can be used to access all information about the data source, including its application operating status, transaction processing status, and error information.

Prototype

```
SQLRETURN  SQLConnect(SQLHDBC        ConnectionHandle,
                      SQLCHAR        *ServerName,
                      SQLSMALLINT    NameLength1,
                      SQLCHAR        *UserName,
                      SQLSMALLINT    NameLength2,
                      SQLCHAR        *Authentication,
                      SQLSMALLINT    NameLength3);
```

Parameter

Table 5-38 SQLConnect parameters

Keyword	Description
ConnectionHandle	Connection handle, obtained from SQLAllocHandle.
ServerName	Name of the data source to connect.
NameLength1	Length of **ServerName**.
UserName	Username of the database in the data source.
NameLength2	Length of **UserName**.
Authentication	User password of the database in the data source.
NameLength3	Length of **Authentication**.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If SQLConnect returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_DBC** and **ConnectionHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

5.4.7.9 SQLDisconnect

Description

Closes the connection associated with a database connection handle.

Prototype

```
SQLRETURN SQLDisconnect(SQLHDBC     ConnectionHandle);
```

Parameter

Table 5-39 SQLDisconnect parameter

Keyword	Description
ConnectionHandle	Connection handle, obtained from SQLAllocHandle.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.

Precautions

If SQLDisconnect returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_DBC** and **ConnectionHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

5.4.7.10 SQLExecDirect

Description

Executes a prepared statement specified in this parameter. SQLExecDirect is the fastest method for executing only one SQL statement at a time.

Prototype

```
SQLRETURN SQLExecDirect(SQLHSTMT         StatementHandle,
                        SQLCHAR          *StatementText,
                        SQLINTEGER       TextLength);
```

Parameters

Table 5-40 SQLExecDirect parameters

Keyword	Description
StatementHandle	Statement handle, obtained from SQLAllocHandle.
StatementText	SQL statement to be executed. Multiple statements cannot be executed at a time.
TextLength	Length of **StatementText**.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_NEED_DATA** indicates that parameters provided before executing the SQL statement are insufficient.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.
- **SQL_NO_DATA** indicates that the SQL statement does not return a result set.

Precautions

If SQLExecDirect returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

5.4.7.11 SQLExecute

Description

If a statement contains a parameter marker, the SQLExecute function uses the current value of the parameter marker to execute a prepared SQL statement.

Prototype

```
SQLRETURN SQLExecute(SQLHSTMT    StatementHandle);
```

Parameter

Table 5-41 SQLExecute parameter

Keyword	Description
StatementHandle	Statement handle to be executed.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_NEED_DATA** indicates that parameters provided before executing the SQL statement are insufficient.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_NO_DATA** indicates that the SQL statement does not return a result set.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If SQLExecute returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

5.4.7.12 SQLFetch

Description

Advances the cursor to the next row of the result set and retrieves any bound columns.

Prototype

```
SQLRETURN SQLFetch(SQLHSTMT    StatementHandle);
```

Parameter

Table 5-42 SQLFetch parameter

Keyword	Description
StatementHandle	Statement handle, obtained from SQLAllocHandle.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_NO_DATA** indicates that the SQL statement does not return a result set.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If SQLFetch returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

5.4.7.13 SQLFreeStmt

In ODBC 3.x, SQLFreeStmt (an ODBC 2.x function) was deprecated and replaced by SQLFreeHandle. For details, see **SQLFreeHandle**.

5.4.7.14 SQLFreeConnect

In ODBC 3.x, SQLFreeConnect (an ODBC 2.x function) was deprecated and replaced by SQLFreeHandle. For details, see **SQLFreeHandle**.

5.4.7.15 SQLFreeHandle

Description

Releases resources associated with a specific environment, connection, or statement handle. It replaces the ODBC 2.x functions: SQLFreeEnv, SQLFreeConnect, and SQLFreeStmt.

Prototype

```
SQLRETURN SQLFreeHandle(SQLSMALLINT   HandleType,
                        SQLHANDLE     Handle);
```

Parameter

Table 5-43 SQLFreeHandle parameters

Keyword	Description
HandleType	Type of handle to be freed by SQLFreeHandle. The value must be one of the following: • **SQL_HANDLE_ENV** • **SQL_HANDLE_DBC** • **SQL_HANDLE_STMT** • **SQL_HANDLE_DESC** If none of them is the value of **HandleType**, SQLFreeHandle returns **SQL_INVALID_HANDLE**.
Handle	Name of the handle to be freed.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.

Precautions

If SQLFreeHandle returns **SQL_ERROR**, the handle is still valid.

Examples

See **Examples**.

5.4.7.16 SQLFreeEnv

In ODBC 3.x, SQLFreeEnv (an ODBC 2.x function) was deprecated and replaced by SQLFreeHandle. For details, see **SQLFreeHandle**.

5.4.7.17 SQLPrepare

Description

Prepares an SQL statement to be executed.

Note that the prepared statements sent by ODBC do not support the kernel reuse plan. As a result, a new plan needs to be generated for each execution, causing high CPU usage. If services have requirements on plan reuse, you are advised to use the JDBC client.

Prototype

```
SQLRETURN SQLPrepare(SQLHSTMT      StatementHandle,
                     SQLCHAR       *StatementText,
                     SQLINTEGER    TextLength);
```

Parameter

Table 5-44 SQLPrepare parameters

Keyword	Description
StatementHandle	Statement handle.
StatementText	SQL text string.
TextLength	Length of **StatementText**.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If SQLPrepare returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

5.4.7.18 SQLGetData

Description

SQLGetData is used to retrieve data for a single column in the result set. It can be called multiple times to partially retrieve variable-length data.

Prototype

```
SQLRETURN SQLGetData(SQLHSTMT        StatementHandle,
                     SQLUSMALLINT    Col_or_Param_Num,
                     SQLSMALLINT     TargetType,
                     SQLPOINTER      TargetValuePtr,
                     SQLLEN          BufferLength,
                     SQLLEN          *StrLen_or_IndPtr);
```

Parameter

Table 5-45 SQLGetData parameters

Keyword	Description
StatementHandle	Statement handle, obtained from SQLAllocHandle.
Col_or_Param_Num	Column number for which the data retrieval is requested. The column number starts with 1 and increases in ascending order. The number of the bookmark column is 0.
TargetType	C data type in the TargetValuePtr buffer. If **TargetType** is **SQL_ARD_TYPE**, the driver uses the data type of the **SQL_DESC_CONCISE_TYPE** field in ARD. If it is **SQL_C_DEFAULT**, the driver selects a default data type according to the source SQL data type.
TargetValuePtr	**Output parameter**: pointer to the pointer that points to the buffer where the data is located.
BufferLength	Size of the buffer pointed to by **TargetValuePtr**.
StrLen_or_IndPtr	**Output parameter**: pointer to the buffer where the length or identifier value is returned.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_NO_DATA** indicates that the SQL statement does not return a result set.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If SQLGetData returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_STMT** and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

5.4.7.19 SQLGetDiagRec

Description

Returns the current values of multiple fields in a diagnostic record that contains error, warning, and status information.

Prototype

```
SQLRETURN  SQLGetDiagRec(SQLSMALLINT    HandleType
                         SQLHANDLE      Handle,
                         SQLSMALLINT    RecNumber,
                         SQLCHAR        *SQLState,
                         SQLINTEGER     *NativeErrorPtr,
                         SQLCHAR        *MessageText,
                         SQLSMALLINT    BufferLength
                         SQLSMALLINT    *TextLengthPtr);
```


Parameters

Table 5-46 SQLGetDiagRec parameters

Keyword	Description
HandleType	Handle-type identifier that describes the type of handle for which diagnostics are desired. The value must be one of the following: • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC
Handle	Handle for the diagnostic data structure. Its type is indicated by **HandleType**. If **HandleType** is set to **SQL_HANDLE_ENV**, **Handle** may indicate a shared or non-shared environment handle.
RecNumber	Status record from which the application seeks information. **RecNumber** starts with 1.
SQLState	**Output parameter**: pointer to a buffer that saves the 5-character **SQLSTATE** code pertaining to **RecNumber**.
NativeErrorPtr	**Output parameter**: pointer to a buffer that saves the native error code.
MessageText	Pointer to a buffer that saves text strings of diagnostic information.
BufferLength	Length of **MessageText**.
TextLengthPtr	**Output parameter**: pointer to the buffer, the total number of bytes in the returned **MessageText**. If the number of bytes available to return is greater than **BufferLength**, then the diagnostics information text in **MessageText** is truncated to **BufferLength** minus the length of the null termination character.

Return Value

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.

Precautions

SQLGetDiagRec does not release diagnostic records for itself. It uses the following return values to report execution results:

- **SQL_SUCCESS** indicates that the function successfully returns diagnostic information.
- **SQL_SUCCESS_WITH_INFO** indicates that the **MessageText** buffer is too small to hold the requested diagnostic information. No diagnostic records are generated.
- **SQL_INVALID_HANDLE** indicates that the handle indicated by **HandleType** and **Handle** is an invalid handle.
- **SQL_ERROR** indicates that the value of **RecNumber** is less than or equal to **0** or the value of **BufferLength** is less than **0**.

If an ODBC function returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, you can call SQLGetDiagRec to obtain the **SQLSTATE** value. **Table 2 SQLSTATE values** lists the possible **SQLSTATE** values.

Table 5-47 SQLSTATE values

SQLSATATE	Error	Description
HY000	General error.	An error occurs when there is no specific SQLSTATE.
HY001	Memory allocation error.	The driver is unable to allocate memory required to support execution or completion of the function.
HY008	Operation canceled.	SQLCancel is called to terminate the statement execution, but the StatementHandle function is still called.
HY010	Function sequence error.	The function is called prior to sending data to data parameters or columns being executed.
HY013	Memory management error.	The function fails to be called. The error may be caused by low memory conditions.
HYT01	Connection timeout.	The timeout period expired before the application was able to connect to the data source.
IM001	Function not supported by the driver.	The called function is not supported by the StatementHandle driver.

Examples

See **Examples**.

5.4.7.20 SQLSetConnectAttr

Description

Sets connection attributes.

Prototype

```
SQLRETURN SQLSetConnectAttr(SQLHDBC      ConnectionHandle
                            SQLINTEGER   Attribute,
                            SQLPOINTER   ValuePtr,
                            SQLINTEGER   StringLength);
```

Parameters

Table 5-48 SQLSetConnectAttr parameters

Keyword	Description
ConnectionHandle	Connection handle.
Attribute	Attribute to set.
ValuePtr	Pointer to the **Attribute** value. **ValuePtr** depends on the **Attribute** value, and can be a 32-bit unsigned integer value or a null-terminated string. If the **ValuePtr** parameter is driver-specific value, it may be a signed integer.
StringLength	If **ValuePtr** points to a string or a binary buffer, **StringLength** is the length of ***ValuePtr**. If **ValuePtr** points to an integer, **StringLength** is ignored.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.

Precautions

If SQLSetConnectAttr returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to **SQL_HANDLE_DBC** and **ConnectionHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

5.4.7.21 SQLSetEnvAttr

Description

Sets environment attributes.

Prototype

```
SQLRETURN SQLSetEnvAttr(SQLHENV      EnvironmentHandle
                        SQLINTEGER   Attribute,
                        SQLPOINTER   ValuePtr,
                        SQLINTEGER   StringLength);
```

Parameter

Table 5-49 SQLSetEnvAttr parameters

Keyword	Description
EnvironmentHandle	Environment handle.
Attribute	Environment attribute to be set. The value must be one of the following: • **SQL_ATTR_ODBC_VERSION**: ODBC version • **SQL_CONNECTION_POOLING**: connection pool attribute • **SQL_OUTPUT_NTS**: string type returned by the driver
ValuePtr	Pointer to the **Attribute** value. **ValuePtr** depends on the **Attribute** value, and can be a 32-bit integer value or a null-terminated string.
StringLength	If **ValuePtr** points to a string or a binary buffer, **StringLength** is the length of ***ValuePtr**. If **ValuePtr** points to an integer, **StringLength** is ignored.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.

Precautions

If SQLSetEnvAttr returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_ENV** and **EnvironmentHandle**, and obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

5.4.7.22 SQLSetStmtAttr

Description

Sets attributes related to a statement.

Prototype

```
SQLRETURN SQLSetStmtAttr(SQLHSTMT      StatementHandle
                         SQLINTEGER    Attribute,
                         SQLPOINTER    ValuePtr,
                         SQLINTEGER    StringLength);
```

Parameter

Table 5-50 SQLSetStmtAttr parameters

Keyword	Description
StatementHandle	Statement handle.
Attribute	Attribute to set.
ValuePtr	Pointer to the **Attribute** value. **ValuePtr** depends on the **Attribute** value, and can be a 32-bit unsigned integer value or a pointer to a null-terminated string, a binary buffer, or a driver-specified value. If the **ValuePtr** parameter is driver-specific value, it may be a signed integer.
StringLength	If **ValuePtr** points to a string or a binary buffer, **StringLength** is the length of ***ValuePtr**. If **ValuePtr** points to an integer, **StringLength** is ignored.

Return Values

- **SQL_SUCCESS** indicates that the call succeeded.
- **SQL_SUCCESS_WITH_INFO** indicates that some warning information is displayed.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. This value may also be returned by other APIs.

Precautions

If SQLSetStmtAttr returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can call **SQLGetDiagRec**, with **HandleType** and **Handle** set to

SQL_HANDLE_STMT and **StatementHandle**, respectively, to obtain the **SQLSTATE** value. The **SQLSTATE** value provides the detailed function calling information.

Examples

See **Examples**.

5.5 Development Based on libpq

libpq is a C application programming interface to GaussDB. libpq contains a set of library functions that allow client programs to send query requests to the GaussDB servers and obtain query results. It is also the underlying engine of other GaussDB APIs, such as ODBC. This chapter provides examples to show how to write code using libpq.

5.5.1 libpq Package, Dependent Library, and Header File

Obtain the libpq package, dependent library, and header files from the release package **GaussDB-Kernel**_*Database version number*_*OS version number*_**64bit_Libpq.tar.gz**. Client programs that use libpq must include the header file **libpq-fe.h** and must connect to the libpq library.

5.5.2 Development Process

To compile and develop source programs based on libpq, perform the following steps:

1. Decompress the **GaussDB-Kernel**_*Database version number*_*OS version number*_**64bit_Libpq.tar.gz** file. The required header file is stored in the **include** folder, and the **lib** folder contains the required libpq library file.

   NOTE

   In addition to **libpq-fe.h**, the **include** folder contains the header files **postgres_ext.h**, **gs_thread.h**, and **gs_threadlocal.h** by default. These three header files are the dependency files of **libpq-fe.h**.

2. Develop the source program **testlibpq.c**. The source code file needs to reference the header file provided by libpq.
   Example: #include <libpq-fe.h>

3. To compile the libpq source program by running **gcc**, use the **-I** *directory* option to provide the installation location of the header file. (Sometimes the compiler looks for the default directory, so this option can be ignored.)
   Example:
   gcc -I *(Directory where the header file is located)* -L *(Directory where the libpq library is located)* testlibpq.c -lpq

4. If the makefile is used, add the following option to variables *CPPFLAGS*, *LDFLAGS*, and *LIBS*:
   CPPFLAGS += -I *(Directory of the header file)*
   LDFLAGS += -L *(Directory of the libpq library)*
   LIBS += -lpq
   For example:
   CPPFLAGS += -I$(GAUSSHOME)/include/libpq
   LDFLAGS += -L$(GAUSSHOME)/lib

5.5.3 Example

Code for Common Functions

Example 1:

```
/*
 * testlibpq.c
* Note: testlibpq.c source program provides basic and common application scenarios of libpq.
* The PQconnectdb, PQexec, PQntuples, and PQfinish interfaces provided by libpq are used to establish database connections, execute SQL statements, obtain returned results, and clear resources.
 */
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>
#include <string.h>

static void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int
main(int argc, char **argv)
{
    /* The values of variables such as user and passwd must be read from environment variables or configuration files. Environment variables need to be configured as required. If no environment variable is used, a character string can be directly assigned. */
    const char conninfo[1024];
    PGconn     *conn;
    PGresult   *res;
    int         nFields;
    int         i,j;
    char        *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char        *port = getenv("EXAMPLE_PORT_ENV");
    char        *host = getenv("EXAMPLE_HOST_ENV");
    char        *username = getenv("EXAMPLE_USERNAME_ENV");
    char        *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /*
     * This value is used when the user provides the value of the conninfo character string in the command line.
     * Otherwise, the environment variables or the default values
     * are used for all other connection parameters.
     */
    if (argc > 1)
        conninfo = argv[1];
    else
        sprintf(conninfo,
            "dbname=%s port=%s host=%s application_name=test connect_timeout=5 sslmode=allow user=%s password=%s",
            dbname, port, host, username, passwd);

    /* Connect to the database. */
    conn = PQconnectdb(conninfo);

    /* Check whether the backend connection has been successfully established. */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
                PQerrorMessage(conn));
        exit_nicely(conn);
    }

    /*
     * Since a cursor is used in the test case, a transaction block is required.
```



```
    * Put all data in one "select * from pg_database"
    * PQexec() is too simple and is not recommended.
    */

    /* Start a transaction block. */
    res = PQexec(conn, "BEGIN");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "BEGIN command failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    /*
    * PQclear PGresult should be executed when it is no longer needed, to avoid memory leakage.
    */
    PQclear(res);

    /*
    * Fetch data from the pg_database system catalog.
    */
    res = PQexec(conn, "DECLARE myportal CURSOR FOR select * from pg_database");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "DECLARE CURSOR failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }
    PQclear(res);

    res = PQexec(conn, "FETCH ALL in myportal");
    if (PQresultStatus(res) != PGRES_TUPLES_OK)
    {
        fprintf(stderr, "FETCH ALL failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    /* First, print out the attribute name. */
    nFields = PQnfields(res);
    for (i = 0; i < nFields; i++)
        printf("%-15s", PQfname(res, i));
    printf("\n\n");

    /* Print lines. */
    for (i = 0; i < PQntuples(res); i++)
    {
        for (j = 0; j < nFields; j++)
            printf("%-15s", PQgetvalue(res, i, j));
        printf("\n");
    }

    PQclear(res);

    /* Close the portal. We do not need to check for errors. */
    res = PQexec(conn, "CLOSE myportal");
    PQclear(res);

    /* End the transaction. */
    res = PQexec(conn, "END");
    PQclear(res);

    /* Close the database connection and clean up the database. */
    PQfinish(conn);

    return 0;
}
```

Example 2:


```
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>
#include <string.h>
/* Test the automatic primary node selection of the PQconnectStart + loop PQconnectStart APIs when
multiple IP addresses are configured. */

static void exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int main()
{
    char *conninfo = "postgresql://10.442.13.173:53600,10.442.13.177:54600/postgres?
user=bot&password=Gaussdba@Mpp&target_session_attrs=read-write";
    //PGconn *conn = PQconnectdb(conninfo);
    PGconn *conn = PQconnectStart(conninfo);
    if (conn == NULL) {
        fprintf(stderr, "Connection initialization failed\n");
        return 1;
    }

    ConnStatusType status;
    int sock;

    while (1) {
        PQconnectPoll(conn);
        status = PQstatus(conn);
        fprintf(stderr, "\n--------conn->status: %d\n", PQstatus(conn));
        if (status == CONNECTION_BAD) {
            fprintf(stderr, "Connection failed\n");
            PQfinish(conn);
            return 1;
        }

        if ((int)status == 0) {
            fprintf(stderr, "Connection established\n");
            break;
        }

        /* Obtain the bottom-layer socket descriptor to perform non-blocking I/O operations. */
        sock = PQsocket(conn);

        /* Other non-blocking operations can be performed here, for example, waiting for an event. */

        /* Simulate other operations in the non-blocking process by using sleep. */
        sleep(1);
    }
    fprintf(stderr, "\n--------conn->status: %d\n", PQstatus(conn));

    PGresult *res;
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s", PQerrorMessage(conn));
        exit_nicely(conn);
    }

    res = PQexec(conn, "show transaction_read_only;");
    printf("\n----------trancaction_read_only is %-15s\n", PQgetvalue(res, 0, 0));
    PQclear(res);

    res = PQexec(conn, "select 1;");
    if (PQresultStatus(res) != PGRES_TUPLES_OK)
    {
        fprintf(stderr, "select failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
```



```
    }
    printf("%-15s\n", PQgetvalue(res, 0, 0));
    PQclear(res);

    PQfinish(conn);
    return 0;
}
```

Example 3:

```
/*
* testlibpq3.c Test PQprepare
* PQprepare creates a prepared statement with specified parameters for PQexecPrepared to execute the
prepared statement.
 * Before running this example, run the following command to create a table:
 * create table t01(a int, b int);
 * insert into t01 values(1, 23);
 */
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>
#include <string.h>
int main(int argc, char * argv[])
{
    /* The values of variables such as user and passwd must be read from environment variables or
configuration files. Environment variables need to be configured as required. If no environment variable is
used, a character string can be directly assigned. */
    PGconn *conn;
    PGresult * res;
    ConnStatusType pgstatus;
    char connstr[1024];
    char cmd_sql[2048];
    int nParams = 0;
    int paramLengths[5];
    int paramFormats[5];
    Oid paramTypes[5];
    char * paramValues[5];
    int i, cnt;
    char cid[32];
    int k;
    char *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char *port = getenv("EXAMPLE_PORT_ENV");
    char *hostaddr = getenv("EXAMPLE_HOST_ENV");
    char *username = getenv("EXAMPLE_USERNAME_ENV");
    char *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /* Use PQconnectdb to connect to the database. The detailed connection information is as follows:
connstr */
    sprintf(connstr,
            "hostaddr=%s dbname=%s port=%s user=%s password=%s",
             hostaddr, dbname, port, username, paswswd);
    conn = PQconnectdb(connstr);
    pgstatus = PQstatus(conn);
    if (pgstatus == CONNECTION_OK)
    {
        printf("Connect database success!\n");
    }
    else
    {
        printf("Connect database fail:%s\n",PQerrorMessage(conn));
        return -1;
    }

    /* Create table t01. */
    res = PQexec(conn, "DROP TABLE IF EXISTS t01;CREATE TABLE t01(a int, b int);INSERT INTO t01
values(1, 23);");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        printf("Command failed: %s.\n", PQerrorMessage(conn));
        PQfinish(conn);
```



```
    return -1;
  }

  /* cmd_sql query */
  sprintf(cmd_sql, "SELECT b FROM t01 WHERE a = $1");
  /*Parameter corresponding to $1 in cmd_sql*/
  paramTypes[0] = 23;
  /* PQprepare creates a prepared statement with given parameters. */
  res = PQprepare(conn,
                  "pre_name",
                  cmd_sql,
                  1,
                  paramTypes);
 if( PQresultStatus(res) != PGRES_COMMAND_OK )
  {
    printf("Failed to prepare SQL : %s\n: %s\n",cmd_sql, PQerrorMessage(conn));
    PQfinish(conn);
    return -1;
  }
  PQclear(res);
  paramValues[0] = cid;
  for (k=0; k<2; k++)
  {
    sprintf(cid, "%d", 1);
    paramLengths[0] = 6;
    paramFormats[0] = 0;
  /*Execute the prepared statement.*/
    res = PQexecPrepared(conn,
                         "pre_name",
                         1,
                         paramValues,
                         paramLengths,
                         paramFormats,
                         0);
    if( (PQresultStatus(res) != PGRES_COMMAND_OK ) && (PQresultStatus(res) != PGRES_TUPLES_OK))
    {
      printf("%s\n",PQerrorMessage(conn));
      PQclear(res);
      PQfinish(conn);
      return -1;
    }
    cnt = PQntuples(res);
    printf("return %d rows\n", cnt);
    for (i=0; i<cnt; i++)
    {
      printf("row %d: %s\n", i, PQgetvalue(res, i, 0));
    }
    PQclear(res);
  }
  /* The execution is complete. Close the connection. */
  PQfinish(conn);
  return 0;
}
```

Example 4:

```
/*
 * testlibpq4.c
 * Test PQexecParams.
 * PQexecParams executes a command with parameters and requests the query result in binary format.
 * Before running this example, run the following command to populate a database:
 *
 *
 * CREATE TABLE test1 (i int4, t text);
 *
  * INSERT INTO test1 values (2, 'ho there');
 *
 * The expected output is as follows:
 *
 *
```



```
 * tuple 0: got
 *  i = (4 bytes) 2
 *  t = (8 bytes) 'ho there'
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <libpq-fe.h>

/* for ntohl/htonl */
#include <netinet/in.h>
#include <arpa/inet.h>

static void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

/*
 * This function is used to print out the query results. The results are in binary format
* and fetched from the table created in the comment above.
 */
static void
show_binary_results(PGresult *res)
{
    int         i;
    int         i_fnum,
                t_fnum;

    /* Use PQfnumber to avoid assumptions about field order in the result. */
    i_fnum = PQfnumber(res, "i");
    t_fnum = PQfnumber(res, "t");

    for (i = 0; i < PQntuples(res); i++)
    {
        char       *iptr;
        char       *tptr;
        int         ival;

        /* Obtain the field value. (Ignore the possibility that they may be null.) */
        iptr = PQgetvalue(res, i, i_fnum);
        tptr = PQgetvalue(res, i, t_fnum);

        /*
         * The binary representation of INT4 is the network byte order,
         * which is better to be replaced with the local byte order.
         */
        ival = ntohl(*((uint32_t *) iptr));

        /*
         * The binary representation of TEXT is text. Since libpq can append a zero byte to it,
         *  and think of it as a C string.
         *
         */

        printf("tuple %d: got\n", i);
        printf(" i = (%d bytes) %d\n",
               PQgetlength(res, i, i_fnum), ival);
        printf(" t = (%d bytes) '%s'\n",
               PQgetlength(res, i, t_fnum), tptr);
        printf("\n\n");
    }
}

int
```



```
main(int argc, char **argv)
{
    /* The values of variables such as user and passwd must be read from environment variables or
configuration files. Environment variables need to be configured as required. If no environment variable is
used, a character string can be directly assigned. */
    const char conninfo[1024];
    PGconn     *conn;
    PGresult   *res;
    const char *paramValues[1];
    int         paramLengths[1];
    int         paramFormats[1];
    uint32_t    binaryIntVal;
    char        *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char        *port = getenv("EXAMPLE_PORT_ENV");
    char        *hostaddr = getenv("EXAMPLE_HOST_ENV");
    char        *username = getenv("EXAMPLE_USERNAME_ENV");
    char        *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /*
     * If the user provides a parameter on the command line,
     * The value of this parameter is a conninfo character string. Otherwise,
    * Use environment variables or default values.
     */
    if (argc > 1)
        conninfo = argv[1];
    else
        sprintf(conninfo,
            "dbname=%s port=%s host=%s application_name=test connect_timeout=5 sslmode=allow user=%s
password=%s",
            dbname, port, hostaddr, username, passwd);

    /* Connect to the database. */
    conn = PQconnectdb(conninfo);

     /* Check whether the connection to the server was successfully established. */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
                PQerrorMessage(conn));
        exit_nicely(conn);
    }

    res = PQexec(conn, "drop table if exists test1;CREATE TABLE test1 (i int4, t text);");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "command failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    PQclear(res);

    res = PQexec(conn, "INSERT INTO test1 values (2, 'ho there');");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "command failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    PQclear(res);

    /* Convert the integer value "2" to the network byte order. */
    binaryIntVal = htonl((uint32_t) 2);

    /* Set the parameter array for PQexecParams. */
    paramValues[0] = (char *) &binaryIntVal;
    paramLengths[0] = sizeof(binaryIntVal);
    paramFormats[0] = 1;        /* Binary */
```



```
    /* PQexecParams executes a command with parameters. */
    res = PQexecParams(conn,
                       "SELECT * FROM test1 WHERE i = $1::int4",
                       1,       /* One parameter */
                       NULL,    /* Enable the backend to deduce the parameter type. */
                       paramValues,
                       paramLengths,
                       paramFormats,
                       1);      /* Binary result is required. */

    if (PQresultStatus(res) != PGRES_TUPLES_OK)
    {
        fprintf(stderr, "SELECT failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }
    /* Output the binary result.*/
    show_binary_results(res);

    PQclear(res);

    /* Close the database connection and clean up the database. */
    PQfinish(conn);

    return 0;
}
```

5.5.4 libpq Interface Reference

NOTICE

Calls to the PQfn interface using libpq is not supported yet.

5.5.4.1 Database Connection Control Functions

There are database connection control functions for controlling the connections to database servers. An application can connect to multiple servers at a time. For example, a client connects to multiple databases. Each connection is represented by a PGconn object, which is obtained from the function PQconnectdb, PQconnectdbParams, or PQsetdbLogin. You can also obtain a connection object by using the PQconnectStart API together with the asynchronous PQconnectPoll polling method. Note that these functions will always return a non-null object pointer, unless memory allocation fails. The API for establishing a connection is stored in the PGconn object. The PQstatus function can be called to check the return value for a successful connection. The PGconn object stores the SSL context in a local thread. Therefore, the PGconn released thread must be the same as the allocated thread.

5.5.4.1.1 PQconnectdbParams

Description

Establishes a new connection with the database server.

Prototype

```
PGconn* PQconnectdbParams(const char* const* keywords, const char* const* values, int expand_dbname);
```


Parameter

Table 5-51 PQconnectdbParams parameters

Keyword	Description
keywords	An array of strings, each of which is a keyword.
values	Value assigned to each keyword.
expand_dbname	When **expand_dbname** is non-zero, the **dbname** keyword value can be recognized as a connection string. Only **dbname** that first appears is treated in this way, and any subsequent **dbname** value is treated as a database name.

Return Values

PGconn * points to the object pointer that contains a connection. The memory is allocated by the function internally.

Precautions

This function establishes a new database connection using the parameters taken from two NULL-terminated arrays. Unlike PQsetdbLogin, the parameter set can be extended without changing the function signature. Therefore, use of this function (or its non-blocking analogs PQconnectStartParams and PQconnectPoll) is preferred for new application programming.

5.5.4.1.2 PQconnectdb

Description

Establishes a new connection with the database server.

Prototype

```
PGconn* PQconnectdb(const char* conninfo);
```

Parameter

Table 5-52 PQconnectdb parameter

Keyword	Description
conninfo	Connection string. For details about the columns in the string, see **Connection Parameters**.

Return Values

PGconn * points to the object pointer that contains a connection. The memory is allocated by the function internally.

Precautions

- This function establishes a new database connection using the parameters taken from the string **conninfo**.
- The input parameter can be empty, indicating that all default parameters can be used. It can contain one or more values separated by spaces or contain a URL.

Example

For details, see **Example**.

5.5.4.1.3 PQconninfoParse

Description

Returns parsed connection options based on the connection.

Prototype

```
PQconninfoOption* PQconninfoParse(const char* conninfo, char** errmsg);
```

Parameters

Table 5-53

Keyword	Description
conninfo	Passed string. This parameter can be left empty. In this case, the default value is used. It can contain one or more values separated by spaces or contain a URL.
errmsg	Error information.

Return Value

PQconninfoOption pointers

5.5.4.1.4 PQconnectStart

Description

Establishes a non-blocking connection with the database server.

Prototype

```
PGconn* PQconnectStart(const char* conninfo);
```


Parameter

Table 5-54

Keyword	Description
conninfo	String of connection information. This parameter can be left empty. In this case, the default value is used. It can contain one or more values separated by spaces or contain a URL.

Return Values

PGconn pointers

NOTE

- When the non-blocking mode (PQconnectStart+PQconnectPoll) is used to connect to the database and the connection information contains multiple IP addresses, libpq attempts to connect to each IP address in sequence and checks **target_session_attrs** until the connection is successful.
- When the database is connected in non-blocking mode, the timeout detection needs to be processed by the upper-layer application.

5.5.4.1.5 PQerrorMessage

Function

Returns error information on a connection.

Prototype

```
char* PQerrorMessage(const PGconn* conn);
```

Parameter

Table 5-55

Keyword	Parameter Description
conn	Connection handle.

Return Value

char pointers

5.5.4.1.6 PQsetdbLogin

Description

Establishes a new connection with the database server.

Prototype

```
PGconn* PQsetdbLogin(const char* pghost, const char* pgport, const char* pgoptions, const char* pgtty,
    const char* dbName, const char* login, const char* pwd);
```

Parameter

Table 5-56 PQsetdbLogin parameters

Keyword	Description
pghost	Name of the host to be connected. For details, see the **host** column described in **Connection Parameters**.
pgport	Port number of the host server. For details, see the **port** field described in **Connection Parameters**.
pgoptions	Command-line options to be sent to the server during running. For details, see the **options** field described in **Connection Parameters**.
pgtty	Ignored. (This option declares the output direction of server logs.)
dbName	Name of the database to be connected. For details, see the **dbname** field described in **Connection Parameters**.
login	Username for connection. For details, see the **user** column described in **Connection Parameters**.
pwd	Password used for authentication during connection. For details, see the **password** field described in **Connection Parameters**.

Return Value

PGconn * points to the object pointer that contains a connection. The memory is allocated by the function internally.

Precautions

- This function is the predecessor of PQconnectdb with a fixed set of parameters. When an undefined parameter is called, its default value is used. Write NULL or an empty string for any one of the fixed parameters that is to be defaulted.
- If the **dbName** value contains an equal sign (=) or a valid prefix in the connection URL, it is taken as a conninfo string and passed to PQconnectdb, and the remaining parameters are consistent with PQconnectdbParams parameters.

5.5.4.1.7 PQfinish

Description

Closes the connection to the server and releases the memory used by the PGconn object.

Prototype

```
void PQfinish(PGconn* conn);
```

Parameter

Table 5-57 PQfinish parameter

Keyword	Description
conn	Points to the object pointer that contains the connection information.

Precautions

If the server connection attempt fails (as indicated by PQstatus), the application should call PQfinish to release the memory used by the PGconn object. The PGconn pointer must not be used again after PQfinish has been called.

Example

For details, see **Example**.

5.5.4.1.8 PQreset

Function

Resets the communication port to the server.

Prototype

```
void PQreset(PGconn* conn);
```

Parameter

Table 5-58 PQreset parameter

Keyword	Parameter Description
conn	Points to the object pointer that contains the connection information.

Precautions

This function will close the connection to the server and attempt to establish a new connection to the same server by using all the parameters previously used. This function is applicable to fault recovery after a connection exception occurs.

5.5.4.1.9 PQstatus

Description

Returns the connection status.

Prototype

```
ConnStatusType PQstatus(const PGconn* conn);
```

Parameter

Table 5-59 PQ status parameter

Keyword	Description
conn	Points to the object pointer that contains the connection information.

Return Value

ConnStatusType indicates the connection status. The enumerated values are as follows:

```
CONNECTION_STARTED
Waiting for the connection to be established.

CONNECTION_MADE
Connection succeeded; waiting to send

CONNECTION_AWAITING_RESPONSE
Waiting for a response from the server.

CONNECTION_AUTH_OK
Authentication received; waiting for backend startup to complete.

CONNECTION_SSL_STARTUP
Negotiating SSL encryption.

CONNECTION_SETENV
Negotiating environment-driven parameter settings.

CONNECTION_OK
Normal connection.

CONNECTION_BAD
Failed connection.
```

Precautions

The connection status can be one of the preceding values. After the asynchronous connection procedure is complete, only two of them, **CONNECTION_OK** and

CONNECTION_BAD, can return. **CONNECTION_OK** indicates that the connection to the database is normal. **CONNECTION_BAD** indicates that the connection to the database fails. Generally, the **CONNECTION_OK** state remains until PQfinish is called. However, a communication failure may cause the connection status to turn to **CONNECTION_BAD** before the connection procedure is complete. In this case, the application can attempt to call PQreset to restore the communication.

Example

For details, see **Example**.

5.5.4.2 Database Statement Execution Functions

After the connection to the database server is successfully established, you can use the functions described in this section to execute SQL queries and commands.

5.5.4.2.1 PQclear

Description

Releases the storage associated with PGresult. Any query result should be released by PQclear when it is no longer needed.

Prototype

```
void PQclear(PGresult* res);
```

Parameters

Table 5-60 PQclear parameter

Keyword	Description
res	Object pointer that contains the query result.

Precautions

PGresult is not automatically released. That is, it does not disappear when a new query is committed or even if you close the connection. To delete it, you must call PQclear. Otherwise, memory leakage occurs.

Example

For details, see **Example**.

5.5.4.2.2 PQexec

Function

Commits a command to the server and waits for the result.

Prototype

```
PGresult* PQexec(PGconn* conn, const char* query);
```

Parameter

Table 5-61 PQexec parameters

Keyword	Parameter Description
conn	Points to the object pointer that contains the connection information.
command	Query string to be executed.

Return Value

PGresult indicates the object pointer that contains the query result.

Precautions

The PQresultStatus function should be called to check the return value for any errors (including the value of a null pointer, in which **PGRES_FATAL_ERROR** will be returned). The PQerrorMessage function can be called to obtain more information about such errors.

NOTICE

The command string can contain multiple SQL commands separated by semicolons (;). Multiple queries sent in a PQexec call are processed in one transaction, unless there are specific BEGIN/COMMIT commands in the query string to divide the string into multiple transactions. Note that the returned PGresult structure describes only the result of the last command executed from the string. If a command fails, the string processing stops and the returned PGresult describes the error condition.

Example

For details, see **Example**.

5.5.4.2.3 PQexecParams

Description

Runs a command to bind parameters.

Prototype

```
PGresult* PQexecParams(PGconn* conn,
                       const char* command,
                       int nParams,
```



```
                const Oid* paramTypes,
                const char* const* paramValues,
                const int* paramLengths,
                const int* paramFormats,
                int resultFormat);
```

Parameter

Table 5-62 PQexecParams parameters

Keyword	Description
conn	Connection handle.
command	SQL text string.
nParams	Number of parameters to be bound.
paramTypes	Types of parameters to be bound.
paramValues	Values of parameters to be bound.
paramLengths	Parameter lengths.
paramFormats	Parameter formats (text or binary).
resultFormat	Result format (text or binary).

Return Value

PGresult pointers

5.5.4.2.4 PQexecParamsBatch

Description

Runs a command to bind batches of parameters.

Prototype

```
PGresult* PQexecParamsBatch(PGconn* conn,
                        const char* command,
                        int nParams,
                        int nBatch,
                        const Oid* paramTypes,
                        const char* const* paramValues,
                        const int* paramLengths,
                        const int* paramFormats,
                        int resultFormat);
```


Parameter

Table 5-63 PQexecParamsBatch parameters

Keyword	Description
conn	Connection handle.
command	SQL text string.
nParams	Number of parameters to be bound.
nBatch	Number of batch operations.
paramTypes	Types of parameters to be bound.
paramValues	Values of parameters to be bound.
paramLengths	Parameter lengths.
paramFormats	Parameter formats (text or binary).
resultFormat	Result format (text or binary).

Return Value

PGresult pointers

5.5.4.2.5 PQexecPrepared

Function

PQexecPrepared is used to send a request to execute a prepared statement with given parameters and wait for the result.

Prototype

```
PGresult* PQexecPrepared(PGconn* conn,
                         const char* stmtName,
                         int nParams,
                         const char* const* paramValues,
                         const int* paramLengths,
                         const int* paramFormats,
                         int resultFormat);
```

Parameter

Table 5-64 PQexecPrepared parameters

Keyword	Parameter Description
conn	Connection handle.

Keyword	Parameter Description
stmtName	*stmt* name, which can be set to "" or NULL to reference an unnamed statement. Otherwise, it must be the name of an existing prepared statement.
nParams	Parameter quantity.
paramValues	Actual values of parameters.
paramLengths	Actual data lengths of parameters.
paramFormats	Parameter formats (text or binary).
resultFormat	Return result format (text or binary).

Return Value

PGresult pointers

5.5.4.2.6 PQexecPreparedBatch

Function

PQexecPreparedBatch is used to send a request to execute a prepared statement with batches of given parameters and wait for the result.

Prototype

```
PGresult* PQexecPreparedBatch(PGconn* conn,
                              const char* stmtName,
                              int nParams,
                              int nBatchCount,
                              const char* const* paramValues,
                              const int* paramLengths,
                              const int* paramFormats,
                              int resultFormat);
```

Parameter

Table 5-65 PQexecPreparedBatch parameters

Keyword	Parameter Description
conn	Connection handle.
stmtName	*stmt* name, which can be set to "" or NULL to reference an unnamed statement. Otherwise, it must be the name of an existing prepared statement.
nParams	Parameter quantity.
nBatchCount	Number of batches.
paramValues	Actual values of parameters.

Keyword	Parameter Description
paramLengths	Actual data lengths of parameters.
paramFormats	Parameter formats (text or binary).
resultFormat	Return result format (text or binary).

Return Value

PGresult pointers

5.5.4.2.7 PQfname

Function

Returns the column name associated with the given column number. Column numbers start from 0. The caller should not release the result directly. The result will be released when the associated PGresult handle is passed to PQclear.

Prototype

```
char* PQfname(const PGresult* res, int field_num);
```

Parameter

Table 5-66 PQfname parameters

Keyword	Parameter Description
res	Operation result handle.
column_number	Number of columns.

Return Value

char pointers

5.5.4.2.8 PQgetvalue

Description

Returns a single threshold for a row of PGresult. Row and column numbers start from 0. The caller should not release the result directly. The result will be released when the associated PGresult handle is passed to PQclear.

Prototype

```
char* PQgetvalue(const PGresult* res, int tup_num, int field_num);
```


Parameter

Table 5-67 PQgetvalue parameters

Keyword	Description
res	Operation result handle.
row_number	Number of rows.
column_number	Number of columns.

Return Value

For data in text format, PQgetvalue returns a null-terminated string representation of the threshold.

For binary data, the value is a binary representation determined by the typsend and typreceive functions of the data type.

If the threshold is empty, an empty string is returned.

5.5.4.2.9 PQnfields

Function

Returns the number of columns (fields) in each row of the query result.

Prototype

```
int PQnfields(const PGresult* res);
```

Parameter

Table 5-68 PQnfields parameters

Keyword	Parameter Description
res	Operation result handle.

Return Value

Value of the int type

5.5.4.2.10 PQntuples

Function

Returns the number of rows (tuples) in the query result.

Prototype

```
int PQntuples(const PGresult* res);
```

Parameter

Table 5-69 PQntuples parameters

Keyword	Parameter Description
res	Operation result handle.

NOTICE

PQntuples returns an integer result. An overflow may occur if the return value is out of the value range allowed in a 32-bit OS.

Return Value

Value of the int type

Examples

For details, see **Example**.

5.5.4.2.11 PQprepare

Description

Commits a request to create a prepared statement with given parameters and waits for completion.

Prototype

```
PGresult* PQprepare(PGconn* conn, const char* stmtName, const char* query, int nParams, const Oid* paramTypes);
```

Parameters

Table 5-70 PQprepare parameters

Keyword	Description
conn	Points to the object pointer that contains the connection information.
stmtName	Name of **prepare** to be executed.
query	Query string to be executed.
nParams	Parameter quantity.

Keyword	Description
paramTypes	Array of the parameter type.

Return Value

PGresult indicates the object pointer that contains the query result.

Precautions

- PQprepare creates a prepared statement for later execution with PQexecPrepared. This function allows commands to be repeatedly executed, without being parsed and planned each time they are executed. PQprepare is supported only in protocol 3.0 or later. It will fail when protocol 2.0 is used.
- This function creates a prepared statement named **stmtName** from the query string, which must contain an SQL command. **stmtName** can be **""** to create an unnamed statement. In this case, any pre-existing unnamed statement will be automatically replaced. Otherwise, this is an error if the statement name has been defined in the current session. If any parameters are used, they are referred to in the query as $1, $2, and so on. **nParams** is the number of parameters for which types are pre-specified in the array paramTypes[]. (The array pointer can be **NULL** when **nParams** is **0**.) paramTypes[] specifies the data types to be assigned to the parameter symbols by OID. If **paramTypes** is **NULL**, or any element in the array is **0**, the server assigns a data type to the parameter symbol in the same way as it does for an untyped literal string. In addition, the query can use parameter symbols whose numbers are greater than **nParams**. Data types of these symbols will also be inferred.

NOTICE

You can also execute the **SQLPREPARE** statement to create a prepared statement that is used with PQexecPrepared. Although there is no libpq function of deleting a prepared statement, the **SQL DEALLOCATE** statement can be used for this purpose.

Example

For details, see **Example**.

5.5.4.2.12 PQresultStatus

Description

Returns the result status of a command.

Prototype

```
ExecStatusType PQresultStatus(const PGresult* res);
```


Parameters

Table 5-71 PQresultStatus parameter

Keyword	Description
res	Object pointer that contains the query result.

Return Value

PQresultStatus indicates the command execution status. The enumerated values are as follows:

```
PQresultStatus can return one of the following values:
PGRES_EMPTY_QUERY
The string sent to the server was empty.

PGRES_COMMAND_OK
A command that does not return data was successfully executed.

PGRES_TUPLES_OK
A query (such as SELECT or SHOW) that returns data was successfully executed.

PGRES_COPY_OUT
The data copied from the server begins to transmit.

PGRES_COPY_IN
The data copied to the server begins to transmit.

PGRES_BAD_RESPONSE
The response from the server cannot be understood.

PGRES_NONFATAL_ERROR
A non-fatal error (notification or warning) occurred.

PGRES_FATAL_ERROR
A fatal error occurred.

PGRES_COPY_BOTH
The data copied to and from the server begins to transmit. This state occurs only in streaming replication.

PGRES_SINGLE_TUPLE
PGresult contains a result tuple from the current command. This state occurs in a single-row query.
```

Precautions

- Note that the SELECT command that happens to retrieve zero rows still returns **PGRES_TUPLES_OK**. **PGRES_COMMAND_OK** is used for commands that can never return rows (such as INSERT or UPDATE, without return clauses). The result status **PGRES_EMPTY_QUERY** might indicate a bug in the client software.
- The result status **PGRES_NONFATAL_ERROR** will never be returned directly by PQexec or other query execution functions. Instead, such results will be passed to the notice processor.

Example

For details, see **Example**.

5.5.4.2.13 PQfnumberEx

Description

Returns the column number associated with the given column name. Columns are numbered starting at 0.

Prototype

```
int PQfnumberEx(const PGresult* res, const char* field_name, bool case_sensitive);
```

Parameters

Table 5-72 PQfnumberEx parameters

Keyword	Description
res	Handle to the operation result.
field_name	Column name.
case_sensitive	Specifies whether the column name is case sensitive.

Return Value

An integer of the int type.

5.5.4.3 Functions for Asynchronous Command Processing

The PQexec function is adequate for committing commands in common, synchronous applications. However, it has several defects, which may be important to some users:

- PQexec waits for the end of the command, but the application may have other work to do (for example, maintaining a user interface). In this case, PQexec would not want to be blocked to wait for the response.
- As the client application is suspended while waiting for the result, it is difficult for the application to determine whether to cancel the ongoing command.
- PQexec can return only one PGresult structure. If the committed command string contains multiple SQL commands, all the PGresult structures except the last PGresult are discarded by PQexec.
- PQexec always collects the entire result of the command and caches it in a PGresult. Although this mode simplifies the error handling logic for applications, it is impractical for results that contain multiple rows.

Applications that do not want to be restricted by these limitations can use the following functions that PQexec is built from: PQsendQuery and PQgetResult. The functions PQsendQueryParams, PQsendPrepare, and PQsendQueryPrepared can also be used with PQgetResult.

5.5.4.3.1 PQsendQuery

Function

Commits a command to the server without waiting for the result. If the query is successful, **1** is returned. Otherwise, **0** is returned.

Prototype

```
int PQsendQuery(PGconn*conn, const char* query);
```

Parameter

Table 5-73 PQsendQuery parameters

Keyword	Parameter Description
conn	Points to the object pointer that contains the connection information.
command	Query string to be executed.

Return Value

int indicates the execution result. **1** indicates successful execution and **0** indicates an execution failure. The failure cause is stored in **conn->errorMessage**.

Precautions

After PQsendQuery is successfully called, call PQgetResult one or more times to obtain the results. PQsendQuery cannot be called again (on the same connection) until PQgetResult returns a null pointer, indicating that the command execution is complete.

5.5.4.3.2 PQsendQueryParams

Function

Commits a command with separated parameters to the server without waiting for the result.

Prototype

```
int PQsendQueryParams(PGconn* conn, const char* command, int nParams, const Oid* paramTypes,
    const char* const* paramValues, const int* paramLengths, const int* paramFormats, int resultFormat);
```


Parameter

Table 5-74 PQsendQueryParams parameters

Keyword	Parameter Description
conn	Points to the object pointer that contains the connection information.
command	Query string to be executed.
nParams	Parameter quantity.
paramTypes	Parameter type.
paramValues	Parameter value.
paramLengths	Parameter length.
paramFormats	Parameter format.
resultFormat	Result format.

Return Value

int indicates the execution result. **1** indicates successful execution and **0** indicates an execution failure. The failure cause is stored in **conn->errorMessage**.

Precautions

This function is equivalent to PQsendQuery. The only difference is that query parameters can be specified separately from the query string. The parameter processing of this function is similar to that of PQexecParams. It cannot work on connections using protocol v2.0, and it allows only one command to appear in the query string.

5.5.4.3.3 PQsendPrepare

Function

Sends a request to create a prepared statement with given parameters, without waiting for completion.

Prototype

```
int PQsendPrepare(PGconn* conn, const char* stmtName, const char* query, int nParams, const Oid* paramTypes);
```


Parameters

Table 5-75 PQsendPrepare parameters

Keyword	Parameter Description
conn	Points to the object pointer that contains the connection information.
stmtName	Name of **prepare** to be executed.
query	Query string to be executed.
nParams	Parameter quantity.
paramTypes	Array of the parameter type.

Return Value

int indicates the execution result. **1** indicates successful execution and **0** indicates an execution failure. The failure cause is stored in **conn->errorMessage**.

Precautions

PQsendPrepare is an asynchronous version of PQprepare. If it can dispatch a request, **1** is returned. Otherwise, **0** is returned. After a successful calling of PQsendPrepare, call PQgetResult to check whether the server successfully created the prepared statement. PQsendPrepare parameters are handled in the same way as PQprepare parameters. Like PQprepare, PQsendPrepare cannot work on connections using protocol 2.0.

5.5.4.3.4 PQsendQueryPrepared

Function

Sends a request to execute a prepared statement with given parameters, without waiting for the result.

Prototype

```
int PQsendQueryPrepared(PGconn* conn, const char* stmtName, int nParams, const char* const*
paramValues,
    const int* paramLengths, const int* paramFormats, int resultFormat);
```

Parameters

Table 5-76 PQsendQueryPrepared parameters

Keyword	Parameter Description
conn	Points to the object pointer that contains the connection information.

Keyword	Parameter Description
stmtName	Name of **prepare** to be executed.
nParams	Parameter quantity.
paramValues	Parameter value.
paramLengths	Parameter length.
paramFormats	Parameter format.
resultFormat	Result format.

Return Value

int indicates the execution result. **1** indicates successful execution and **0** indicates an execution failure. The failure cause is stored in **conn->error_message**.

Precautions

PQsendQueryPrepared is similar to PQsendQueryParams, but the command to be executed is specified by naming a previously-prepared statement, instead of providing a query string. PQsendQueryPrepared parameters are handled in the same way as PQexecPrepared parameters. Like PQexecPrepared, PQsendQueryPrepared cannot work on connections using protocol 2.0.

5.5.4.3.5 PQflush

Description

Flushes any queued output data to the server.

Prototype

```
int PQflush(PGconn* conn);
```

Parameter

Table 5-77 PQflush parameter

Keyword	Description
conn	Points to the object pointer that contains the connection information.

Return Values

int indicates the execution result. If the operation is successful (or the send queue is empty), **0** is returned. If the operation fails, **-1** is returned. If all data in the send

queue fails to be sent, **1** is returned. (This case occurs only when the connection is non-blocking.) The failure cause is stored in **conn->error_message**.

Precautions

Call PQflush after sending any command or data over a non-blocking connection. If **1** is returned, wait for the socket to become read- or write-ready. If the socket becomes write-ready, call PQflush again. If the socket becomes read-ready, call PQconsumeInput and then call PQflush again. Repeat the operation until the value **0** is returned for PQflush. It is necessary to check read-ready and use PQconsumeInput to exhaust input because the server may prevent attempts to send data (such as NOTICE messages) to the client and does not read the client's data until the client reads its data. Once PQflush returns **0**, wait for the socket to be read-ready and then read the response as described above.

5.5.4.4 Functions for Canceling Queries in Progress

A client application can use the functions described in this section to cancel a command that is still being processed by the server.

5.5.4.4.1 PQgetCancel

Description

Creates a data structure that contains the information required to cancel a command issued through a specific database connection.

Prototype

```
PGcancel* PQgetCancel(PGconn* conn);
```

Parameter

Table 5-78 PQgetCancel parameter

Keyword	Description
conn	Points to the object pointer that contains the connection information.

Return Value

PGcancel points to the object pointer that contains the cancel information.

Precautions

PQgetCancel creates a PGcancel object for a given PGconn connection object. If the given connection object (**conn**) is NULL or an invalid connection, it will return NULL. The PGcancel object is an opaque structure that cannot be directly accessed by applications. It can be transferred only to PQcancel or PQfreeCancel.

5.5.4.4.2 PQfreeCancel

Function

Releases the data structure created by PQgetCancel.

Prototype

```
void PQfreeCancel(PGcancel* cancel);
```

Parameter

Table 5-79 PQfreeCancel parameter

Keyword	Parameter Description
cancel	Points to the object pointer that contains the cancel information.

Precautions

PQfreeCancel releases a data object previously created by PQgetCancel.

5.5.4.4.3 PQcancel

Function

Requests the server to abandon processing of the current command.

Prototype

```
int PQcancel(PGcancel* cancel, char* errbuf, int errbufsize);
```

Parameter

Table 5-80 PQcancel parameters

Keyword	Parameter Description
cancel	Points to the object pointer that contains the cancel information.
errbuf	Buffer for storing error information.
errbufsize	Size of the buffer for storing error information.

Return Value

int indicates the execution result. **1** indicates successful execution and **0** indicates an execution failure. The failure cause is stored in **errbuf**.

Precautions

- Successful sending does not guarantee that the request will have any effect. If the cancellation is valid, the current command is terminated early and an error is returned. If the cancellation fails (for example, because the server has processed the command), no result is returned.
- If **errbuf** is a local variable in a signal handler, you can safely call PQcancel from the signal handler. For PQcancel, the PGcancel object is read-only, so it can also be called from a thread that is separate from the thread that is operating the PGconn object.

5.5.5 Connection Parameters

Table 5-81 Connection parameters

Parameter	Description
host	Name of the host to connect to. If the host name starts with a slash (/), UDS communications instead of TCP/IP communications are used. The value is the directory where the socket file is stored. If **host** is not specified, the default behavior is to connect to the UDS in the **/tmp** directory (or the socket directory specified during GaussDB installation). On a machine without a UDS, the default behavior is to connect to **localhost**. You can specify multiple host names by using a character string separated by commas (,). Multiple host names can be specified.

Parameter	Description
hostaddr	IP address of the host to connect to. The value is in standard IPv4 address format, for example, 172.28.40.9. If the machine supports IPv6, IPv6 address can also be used. If a non-null string is specified, TCP/IP communications are used. You can specify multiple IP addresses by using a character string separated by commas (,). Multiple IP addresses can be specified. Replacing **host** with **hostaddr** can prevent applications from querying host names, which may be important for applications with time constraints. However, a host name is required for GSSAPI or SSPI authentication methods. Therefore, the following rules are used: 1. If **host** is specified but **hostaddr** is not, a query for the host name will be executed. 2. If **hostaddr** is specified but **host** is not, the value of **hostaddr** is the server network address. If the host name is required for authentication, the connection attempt fails. 3. If both **host** and **hostaddr** are specified, the value of **hostaddr** is the server network address. The value of **host** is ignored unless it is required by authentication, in which case it is used as the host name. **NOTICE** • If **host** is not the server name in the network address specified by **hostaddr**, the authentication may fail. • If neither **host** nor **hostaddr** is specified, libpq will use a local UDS for connection. If the machine does not have a UDS, it will attempt to connect to **localhost**.
port	Port number of the host server, or the socket file name extension for UDS connections. You can specify multiple port numbers by using a character string separated by commas (,). Multiple port numbers can be specified.
user	Name of the user to be connected. By default, the username is the same as the OS name of the user running the application.
dbname	Database name. The default value is the same as the username.
password	Password to be used if the server requires password authentication.
connect_timeout	Maximum timeout period of the connection, in seconds (written as a decimal integer string). The value **0** or null indicates infinity. You are advised not to set the connection timeout period to a value less than 2 seconds.
client_encoding	Client encoding for the connection. In addition to the values accepted by the corresponding server options, you can use **auto** to determine the correct encoding from the current environment in the client (the *LC_CTYPE* environment variable in the Unix system).

Parameter	Description
tty	This parameter can be ignored. (This parameter was used to specify the location to which the debugging output of the server was sent).
options	Adds command-line options to send to the server at runtime.
application_name	Current user identity.
fallback_application_name	Specifies a backup value for the **application_name** parameter. This value is used if no value is set for **application_name** through a connection parameter or the *PGAPPNAME* environment variable. In a common tool program, if you set a default name but do not want the default name to be overwritten by the user, you can specify a backup value.
keepalives	Specifies whether TCP keepalive is enabled on the client side. The default value is **1**, indicating that the function is enabled. The value **0** indicates that the function is disabled. Ignore this parameter for UDS connections.
keepalives_idle	The number of seconds of inactivity after which TCP should send a keepalive message to the server. The value **0** indicates that the default value is used. Ignore this parameter for UDS connections or if keepalive is disabled.
keepalives_interval	The number of seconds after which a TCP keepalive message that is not acknowledged by the server should be retransmitted. The value **0** indicates that the default value is used. Ignore this parameter for UDS connections or if keepalive is disabled.
keepalives_count	Controls the number of times that keepalive messages are sent through TCP. The value **0** indicates that the default value is used. Ignore this parameter for UDS connections or if keepalive is disabled.
tcp_user_timeout	Specifies the maximum duration for which transmitted data can remain unacknowledged before the TCP connection is forcibly closed on an OS that supports the **TCP_USER_TIMEOUT** socket option. The value **0** indicates that the default value is used. Ignore this parameter for UDS connections.
tcp_syn_retries	Specifies the number of retransmissions due to SYN packet transmission failures in the three-way handshake phase when the client establishes a connection on an OS that supports the **TCP_SYNCNT** socket option. **0** indicates that the default value is used. Ignore this parameter for UDS connections.

Parameter	Description
rw_timeout	Sets the read and write timeout interval of the client connection. When the timeout is triggered on the libpq and the connection is closed, the running services delivered by the libpq to the database are forcibly terminated. This capability is controlled by the GUC parameter **check_disconnect_query**. If this parameter is set to **on**, the capability is supported. If this parameter is set to **off**, the capability is not supported.
sslmode	Specifies whether to enable SSL encryption. ● **disable**: SSL connection is disabled. ● **allow**: If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified. ● **prefer**: If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified. ● **require**: SSL connection is required and data is encrypted. However, authenticity of the database server will not be verified. ● **verify-ca**: SSL connection is required. Currently, Windows ODBC does not support certificate-based authentication. ● **verify-full**: SSL connection is required. Currently, Windows ODBC does not support certificate-based authentication.
sslcompression	If this parameter is set to **1** (default value), the data transmitted over the SSL connection is compressed (this requires that the OpenSSL version be 0.9.8 or later). If this parameter is set to **0**, compression will be disabled (this requires OpenSSL 1.0.0 or later). If a connection without SSL is established, this parameter is ignored. If the OpenSSL version in use does not support this parameter, it will also be ignored. Compression takes up CPU time, but it increases throughput when the bottleneck is the network. If CPU performance is a limiting factor, disabling compression can improve response time and throughput.
sslcert	This parameter specifies the file name of the client SSL certificate. If no SSL connection is established, this parameter is ignored.
sslkey	This parameter specifies the location of the key used for the client certificate. It can specify a key obtained from an external "engine" (the engine is a loadable module of OpenSSL). The description of an external engine should consist of a colon-separated engine name and an engine-related key identifier. If no SSL connection is established, this parameter is ignored.
sslrootcert	This parameter specifies the name of a file that contains the SSL Certificate Authority (CA) certificate. If the file exists, the system authenticates the server certificate issued by one of these authorities.

Parameter	Description
sslcrl	This parameter specifies the file name of the SSL Certificate Revocation List (CRL). If a certificate listed in this file exists, the server certificate authentication will be rejected.
requirepeer	This parameter specifies the OS user of the server, for example, **requirepeer=postgres**. When a UDS connection is established, if this parameter is set, the client checks whether the server process is running under the specified username at the beginning of the connection. If not, the connection will be interrupted by an error. This parameter can be used to provide server authentication similar to that of the SSL certificate on TCP/IP connections. (Note that if the UDS is in **/tmp** or another public writable location, any user can start a server for listening to the location. Use this parameter to ensure that you are connected to a server that is run by a trusted user.) This option is supported only on platforms that implement the peer authentication method.
krbsrvname	This parameter specifies the Kerberos service name used for GSSAPI authentication. For successful Kerberos authentication, this value must match the service name specified in the server configuration.
gsslib	This parameter specifies the GSS library used for GSSAPI authentication. It is used only in the Windows OS. If this parameter is set to **gssapi**, **libpq** is forced to use the GSSAPI library to replace the default SSPI for authentication.
service	This parameter specifies the name of the service for which the additional parameter is used. It specifies a service name in **pg_service.conf** that holds the additional connection parameters. This allows the application to specify only one service name so that the connection parameters can be centrally maintained.
authtype	**authtype** is no longer used, so it is marked as a parameter not to be displayed. It is retained in an array so as not to reject the **conninfo** string from old applications that might still try to set it.
remote_nodename	Specifies the name of the remote node connected to the local node.
localhost	Specifies the local host in a connection channel.
localport	Specifies the local port in a connection channel.
fencedUdfRPCMode	Specifies whether the fenced UDF RPC protocol uses UDSs or special socket file names. The default value is **0**, indicating that the UDS mode is used and the file type is .s.PGSQL.%d. To use the fenced UDF mode, set this parameter to **1**. In this case, the file type is .s.fencedMaster_unixdomain.

Parameter	Description
replication	Specifies whether the connection should use replication protocols instead of common protocols. Protocols with this parameter configured are internal protocols used for PostgreSQL replication connections and tools such as **pg_basebackup**, while they can also be used by third-party applications. The following values, which are case-insensitive, are supported: ● **true**, **on**, **yes**, and **1**: Specifies that the physical replication mode is connected. ● database Specifies that the logical replication mode and the database specified by **dbname** are connected. ● **false**, **off**, **no**, and **0**: Specifies that the connection is a regular connection, which is the default behavior. In physical or logical replication mode, only simple query protocols can be used.
backend_version	Specifies the backend version to be passed to the remote end.
prototype	Sets the current protocol level. The default value is **PROTO_TCP**.
enable_ce	Specifies whether a client is allowed to connect to a fully-encrypted database. The default value is **0**. To enable the basic capability of encrypted equality query, change the value to **1**. To support the memory decryption emergency channel, change the value to **3**.
key_info	This parameter is used together with **enable_ce** to set parameters for accessing an external key manager in an encrypted database.
connection_info	The value of **connection_info** is a JSON character string consisting of **driver_name**, **driver_version**, **driver_path**, and **os_user**. If the value is not NULL, use **connection_info** and ignore **connectionExtraInf**. If the value is null, a connection information string related to **libpq** is generated. When **connectionExtraInf** is set to **false**, the value of **connection_info** consists of only **driver_name** and **driver_version**.
connectionExtraInf	Specifies whether the value of **connection_info** contains extension information. The default value is **0**. If the value contains other information, set this parameter to **1**.

Parameter	Description
target_session_attrs	Specifies the type of the host to be connected. The connection is successful only when the host type is the same as the configured value. This parameter is verified only when multiple IP addresses are specified. The rules for setting **target_session_attrs** are as follows: • **any**: All types of hosts can be connected. • **read-write**: The connection is set up only when the connected host is readable and writable. • **read-only**: Only readable hosts can be connected. • **primary** (default value): Only the primary node in the primary/standby systems can be connected. • **standby**: Only the standby node in the primary/standby systems can be connected. • **prefer-standby**: The system first attempts to find a standby node for connection. If all hosts in the **hosts** list fail to be connected, try the **any** mode.

5.6 Psycopg-based Development

Psycopg is a Python API used to execute SQL statements and provides a unified access API for GaussDB. Applications can perform data operations based on psycopg. Psycopg2 is the encapsulation of libpq and is implemented using the C language, which is efficient and secure. It provides cursors on both clients and servers, asynchronous communication and notification, and the COPY TO and COPY FROM functions. It supports multiple types of Python out-of-the-box and adapts to GaussDB data types. Through the flexible object adaptation system, you can extend and customize the adaptation. Psycopg2 is compatible with Unicode.

GaussDB supports the psycopg2 feature and allows psycopg2 to be connected in SSL mode.

Table 5-82 Platforms supported by psycopg

OS	Platform	Python Version
EulerOS V2.0SP5	• Arm64 • x86_64	3.8.5
EulerOS V2.0SP9	• Arm64 • x86_64	3.7.4
EulerOS V2.0SP10, Kylin V10, and UnionTech20	• Arm64 • x86_64	3.7.9
EulerOS V2.0SP11 and SUSE 12.5	• Arm64 • x86_64	3.9.11

OS	Platform	Python Version
Huawei Cloud EulerOS 2.0	• Arm64 • x86_64	3.9.9

NOTICE

During psycopg2 compilation, OpenSSL of GaussDB is linked. OpenSSL of GaussDB may be incompatible with OpenSSL of the OS. If incompatibility occurs, for example, "version 'OPENSSL_1_1_1f' not found" is displayed, use the environment variable *LD_LIBRARY_PATH* to isolate the OpenSSL provided by the OS and the OpenSSL on which GaussDB depends.

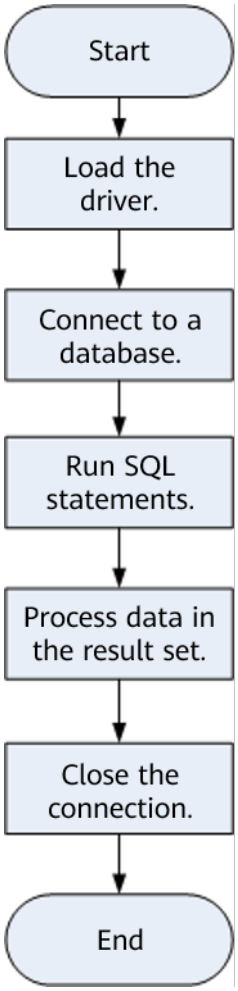
For example, when the application software **client.py** that invokes psycopg2 is executed, the environment variable is explicitly assigned to the application software.

```
export LD_LIBRARY_PATH=/path/to/gaussdb/libs:$LD_LIBRARY_PATH python client.py
```

In the preceding command, **/path/to/pyscopg2/lib** indicates the directory where the OpenSSL library on which the GaussDB depends is located. Change it as required.

5.6.1 Development Process

Figure 5-4 Application development process based on psycopg2



5.6.2 Psycopg Package

Step 1 Prepare related drivers and dependent libraries. Obtain the package **GaussDB-Kernel**_*Database version number*_*OS version number*_**64bit_Python.tar.gz** from the release package.

After the decompression, the following folders are generated:

- **psycopg2**: **psycopg2** library file
- **lib**: **lib** library file

Step 2 Load the driver.

- Before using the driver, perform the following operations:
  a. Decompress the driver package of the corresponding version.
  ```
  tar zxvf xxxx-Python.tar.gz
  ```
  b. Copy **psycopg2** to the **site-packages** folder in the Python installation directory as the **root** user.
  ```
  su root
  cp psycopg2 $(python3 -c 'import site; print(site.getsitepackages()[0])') -r
  ```


c. Change the **psycopg2** directory permission to **755**.
```
chmod 755 $(python3 -c 'import site; print(site.getsitepackages()[0])')/psycopg2 -R
```

d. Add the **psycopg2** directory to the environment variable *$PYTHONPATH* and validate it.
```
export PYTHONPATH=$(python3 -c 'import site; print(site.getsitepackages()[0])'):$PYTHONPATH
```

e. For non-database users, configure the **lib** directory in *LD_LIBRARY_PATH* after decompression.
```
export LD_LIBRARY_PATH=path/to/lib:$LD_LIBRARY_PATH
```

- Load a database driver before creating a database connection.
```
import  psycopg2
```

Step 3 Connect to a database.

Connect to the database in non-SSL mode.

1. Use the psycopg2.connect function to obtain the connection object.
2. Use the connection object to create a cursor object.

Connect to the database in SSL mode.

When you use psycopy2 to connect to the GaussDB server, you can enable SSL to encrypt the communication between the client and server. To enable SSL, you must have the server certificate, client certificate, and private key files. For details on how to obtain these files, see related documents and commands of OpenSSL.

1. Use the .ini file (the **configparser** package of Python can parse this type of configuration file) to save the configuration information about the database connection.
2. Add SSL connection parameters **sslmode**, **sslcert**, **sslkey**, and **sslrootcert** to the connection options.
   a. **sslmode**: For details about the options, see **Table 5-83**.
   b. **sslcert**: client certificate path.
   c. **sslkey**: client key path.
   d. **sslrootcert**: root certificate path.
3. Use the psycopg2.connect function to obtain the connection object.
4. Use the connection object to create a cursor object.

CAUTION

To use SSL to connect to the database, ensure that the Python interpreter is compiled in the mode of generating a dynamic link library (.so) file. You can perform the following steps to check the connection mode of the Python interpreter:

1. Run the **import ssl** command in the Python interpreter to import SSL.
2. Run the **ps ux** command to query the PID of the Python interpreter. Assume that the PID is ******.
3. In the shell CLI, run the **pmap -p ****** | grep ssl** command and check whether the command output contains the path related to **libssl.so**. If yes, the Python interpreter is compiled in dynamic link mode.

Table 5-83 sslmode options

sslmode	Enable SSL Encryption	Description
disable	No	SSL connection is not enabled.
allow	Possible	If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified.
prefer	Possible	If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified.
require	Yes	SSL connection is required, but only data is encrypted. However, authenticity of the database server will not be verified.
verify-ca	Yes	SSL connection is required, and the validity of the server CA must be verified.
verify-full	Yes	The SSL connection must be enabled, which is not supported by GaussDB currently.

Step 4 Run SQL statements.

1. Construct an operation statement and use %s as a placeholder. During execution, psycopg2 will replace the placeholder with the parameter value. You can add the RETURNING clause to obtain the automatically generated column values.
2. Use the cursor.execute method to execute one row of SQL statement, and use the cursor.executemany method to execute multiple rows of SQL statements.

Step 5 Process the result set.

1. cursor.fetchone(): fetches the next row in a query result set and returns a sequence. If no data is available, null is returned.
2. cursor.fetchall(): fetches all remaining rows in a query result and returns a list. An empty list is returned when no rows are available.

NOTE

For database-specific data types, such as tinyint, the corresponding columns in the query result are character strings.

Step 6 Disable the connection.

After you complete required data operations in a database, close the database connection. Call the close method such as connection.close() to close the connection.

CAUTION

This method closes the database connection and does not automatically call commit(). If you just close the database connection without calling commit() first, changes will be lost.

----End

5.6.3 Example: Common Operations

```
import psycopg2
import os

# Obtain the username and password from environment variables.
user = os.getenv('user')
password = os.getenv('password')

# Create a connection object.
conn=psycopg2.connect(database="database", user=user, password=password, host="localhost", port=port)
cur=conn.cursor() # Create a pointer object.

# Create a connection object (using SSL).
conn = psycopg2.connect(dbname="database", user=user, password=password, host="localhost", port=port,
        sslmode="verify-ca", sslcert="client.crt",sslkey="client.key",sslrootcert="cacert.pem")
Note: If sslcert, sslkey, and sslrootcert are not set, the following files in the .postgresql directory of the
current user are used by default: client.crt, client.key, and root.crt.


# Create a table.
cur.execute("CREATE TABLE student(id integer,name varchar,sex varchar);")

# Insert data.
cur.execute("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",(1,'Aspirin','M'))
cur.execute("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",(2,'Taxol','F'))
cur.execute("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",(3,'Dixheral','M'))

# Insert data in batches.
stus = ((4,'John','M'),(5,'Alice','F'),(6,'Peter','M'))
cur.executemany("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",stus)

# Obtain the result.
cur.execute('SELECT * FROM student')
results=cur.fetchall()
print (results)

# Perform a commit.
conn.commit()

# Insert a data record.
cur.execute("INSERT INTO student(id,name,sex) VALUES(%s,%s,%s)",(7,'Lucy','F'))

# Roll back the operation.
conn.rollback()

# Close the connection.
cur.close()
conn.close()

Common connection modes of psycopg2
1. conn = psycopg2.connect(dbname="dbname", user=user, password=password, host="localhost",
port=port)
2. conn = psycopg2.connect(f"dbname=dbname user={user} password={password}  host=localhost
port=port")
3. Use logs.
import logging
import psycopg2
```



```
from psycopg2.extras import LoggingConnection
import os

# Obtain the username and password from environment variables.
user = os.getenv('user')
password = os.getenv('password')

logging.basicConfig(level=logging.DEBUG) # Log level
logger = logging.getLogger(__name__)

db_setting = {
    "user": user,
    "password": password,
    "host": "localhost",
    "database": "dbname",
    "port": port
}

# LoggingConnection records all SQL statements by default. You can filter unnecessary or sensitive SQL statements. The following is an example of filtering password-related SQL statements.
class SelfLoggingConnection(LoggingConnection):

    def filter(self, msg, curs):
        if db_settings['password'] in msg.decode():
            return b'queries containing the password will not be recorded'
        return msg

conn = psycopg2.connect(connection_factory=SelfLoggingConnection, **db_settings)
conn.initialize(logger)
```

CAUTION

- By default, **LoggingConnection** records all SQL information and does not anonymize sensitive information. You can use the filter function to define the output SQL content.
- The log function is an additional function provided by psycopg2 for developers to explicitly debug full SQL statements. By default, the log function is not used. This function prints SQL statements before pyscopg2 executes SQL statements. However, the SQL statements can be printed only when the log level is **DEBUG**. This function is not a default function. It is used only when there are special requirements. You are advised not to use this function unless there are special requirements. For details, visit https://www.psycopg.org/docs/extras.html?highlight=loggingconnection.

5.6.4 Example: Vector Scenario

```
import psycopg2
import os

# Obtain a username and a password from environment variables.
user = os.getenv('user')
password = os.getenv('password')

# Create a connection object.
conn=psycopg2.connect(database="db_test",user=user,password=password,host="localhost",port=port)
cur=conn.cursor() # Create a pointer object.

# Create a connection object (using SSL).
#conn = psycopg2.connect(dbname="database", user=user, password=password, host="localhost", port=port,
#         sslmode="verify-ca", sslcert="client.crt",sslkey="client.key",sslrootcert="cacert.pem")
# Note: If sslcert, sslkey, and sslrootcert are not set, the following files in the .postgresql directory of the current user are used by default: client.crt, client.key, and root.crt.
```



```
# Create a table.
cur.execute("CREATE TABLE IF NOT EXISTS diskann_table (id int, repr FloatVector (128)) with
(storage_type=astore);")

# Import local data in copy mode.
cur.execute("copy diskann_table from '/data/dataSet/sift1b_10w_128.csv' WITH(format 'text', delimiter E'\t',
ignore_extra_data 'true', noescaping 'true');")

# Create an index. Specify the index parameters as required.
cur.execute("set maintenance_work_mem=1024000; CREATE Index diskannidx on diskann_table using
gsdiskann (repr COSINE) with (pq_nseg = 128, pq_nclus = 16, queue_size = 100, enable_pq = false);")

# Insert a data record.
cur.execute("INSERT INTO diskann_table(id,repr) VALUES(%s,%s)",
(0,'[0,16,35,5,32,31,14,10,11,78,55,10,45,83,11,6,14,57,102,75,20,8,3,5,67,17,19,26,5,0,1,22,60,26,7,1,18,22,84,
53,85,119,119,4,24,18,7,7,1,81,106,102,72,30,6,0,9,1,9,119,72,1,4,33,119,29,6,1,0,1,14,52,119,30,3,0,0,55,92,11
1,2,5,4,9,22,89,96,14,1,0,1,82,59,16,20,5,25,14,11,4,0,0,1,26,47,23,4,0,0,4,38,83,30,14,9,4,9,17,23,41,0,0,2,8,19,2
5,23,1]'))

# Insert data in batches.
data =
((900000,'[78,9,0,0,0,0,0,0,163,43,1,5,15,1,0,0,14,4,1,42,75,5,0,0,23,2,7,30,13,0,0,25,83,6,0,0,0,1,0,3,163,22,0,0,
14,9,1,7,38,2,1,21,163,35,3,4,61,43,3,18,29,3,6,20,76,0,0,0,0,1,0,10,163,6,0,0,17,3,0,10,62,1,0,2,163,43,11,7,32,7,
0,1,18,13,43,96,60,0,0,0,0,0,0,0,18,163,1,0,0,12,0,0,33,56,1,0,1,141,6,0,5,97,10,0,0,11,0,2,27]'),
(900001,'[7,7,0,1,9,71,39,6,62,1,0,0,0,19,26,54,119,4,0,0,16,6,7,136,15,0,0,1,136,23,4,14,22,120,11,3,5,39,20,3,7
8,23,5,5,21,44,33,37,136,73,0,0,34,10,8,75,11,5,0,2,136,31,3,5,88,68,6,1,0,1,8,61,80,31,6,2,8,49,40,50,136,46,2,7
,29,5,6,41,19,10,2,32,136,1,0,1,62,31,0,0,0,0,12,51,41,34,3,0,1,5,18,25,83,34,0,5,20,2,2,2,12,8,0,22,136,1,0,0]'))
cur.executemany("INSERT INTO diskann_table(id,repr) VALUES(%s,%s)",data)

# Perform a commit.
conn.commit()

# Query the result.
cur.execute("SELECT id, repr<
+>'[14,35,19,20,3,1,13,11,16,119,85,5,0,5,24,26,0,27,119,13,3,9,19,0,0,11,73,9,10,3,5,0,92,38,17,39,32,7,15,47,1
19,111,53,27,8,0,0,52,5,7,63,51,84,43,0,1,12,8,20,25,33,30,2,5,59,23,25,105,25,23,5,18,119,15,7,13,14,19,95,119
,5,0,0,14,119,103,93,39,11,4,1,4,13,43,62,18,2,0,0,8,44,65,7,1,3,0,0,1,19,45,94,95,13,7,0,0,3,52,119,52,15,2,0,0,0,
11,21,33]' as dist from diskann_table order by dist limit 1;")
results=cur.fetchall()
print (results)

# Close the connection.
cur.close()
conn.close()
```

5.6.5 Psycopg API Reference

Psycopg APIs are a set of methods provided for users. This section describes some common APIs.

5.6.5.1 psycopg2.connect()

Description

This method creates a database session and returns a new connection object.

Prototype

```
import os
conn=psycopg2.connect(dbname="test",user=os.getenv('user'),password=os.getenv('password'),host="127.0.
0.1",port="5432")
```


Parameters

Table 5-84 psycopg2.connect parameters

Keyword	Description
dbname	Database name.
user	Username.
password	Password.
host	IP address of the database. You can specify multiple IP addresses and separate them with commas (,). By default, the UDS is used.
port	Connection port number. The default value is **5432**. If the host has multiple IP addresses and the port numbers are the same, specify one port number. Otherwise, the port numbers must correspond to the IP addresses one by one and are separated by commas (,).
sslmode	SSL mode, which is used for SSL connection.
sslcert	Path of the client certificate, which is used for SSL connection.
sslkey	Path of the client key, which is used for SSL connection.
sslrootcert	Path of the root certificate, which is used for SSL connection.
hostaddr	IP address of the database.
connect_timeout	Client connection timeout interval.
client_encoding	Encoding format of the client.
application_name	Value of **application_name**.
fallback_application_name	Rollback value of **application_name**.
keepalives	Specifies whether to enable the TCP connection on the client. The value can be **1** (enabled) or **0** (disabled). The default value is **1**. If the UDS connection is used, ignore this parameter.
options	Specifies the command line options sent to the server when the connection starts.
keepalives_idle	Describes inactivity before keepalive messages are sent to the server. If keepalive messages are disabled, ignore this parameter.
keepalives_interval	Specifies whether keepalive messages that are not confirmed by the server need to be resent. If keepalive messages are disabled, ignore this parameter.

Keyword	Description
keepalives_count	Specifies the number of TCP connections that may be lost before the client is disconnected from the server.
replication	Specifies that the connection uses the replication protocol instead of the common protocol.
requiressl	Supports the SSL mode.
sslcompression	Specifies the SSL compression. If this parameter is set to **1**, the data sent through the SSL connection is compressed. If this parameter is set to **0**, compression is disabled. If no SSL connection is established, ignore this parameter.
sslcrl	Specifies the path of the CRL, which is used to check whether the SSL server certificate is available.
requirepeer	Specifies the OS username of the server.
target_session_attrs	Specifies the type of the host to be connected. The connection is successful only when the host type is the same as the configured value. This parameter is verified only when multiple IP addresses are specified. The rules for setting **target_session_attrs** are as follows: • **any**: All types of hosts can be connected. • **read-write**: The connection is set up only when the connected host is readable and writable. • **read-only**: Only readable hosts can be connected. • **primary** (default value): Only the primary node in the primary/standby systems can be connected. • **standby**: Only the standby node in the primary/standby systems can be connected. • **prefer-standby**: The system first attempts to find a standby node for connection. If all hosts in the **hosts** list fail to be connected, try the **any** mode.

Return Value

Connection object (for connecting to a database instance)

Examples

For details, see **Example: Common Operations**.

5.6.5.2 connection.cursor()

Function

This method returns a new cursor object.

Prototype

```
cursor(name=None, cursor_factory=None, scrollable=None, withhold=False)
```

Parameter

Table 5-85 connection.cursor parameters

Keyword	Description
name	Cursor name. The default value is **None**.
cursor_factory	Creates a non-standard cursor. The default value is **None**.
scrollable	Sets the SCROLL option. The default value is **None**.
withhold	Sets the HOLD option. The default value is **False**.

Return Value

Cursor object (used for cusors that are programmed using Python in the entire database)

Examples

For details, see **Example: Common Operations**.

5.6.5.3 cursor.execute(query,vars_list)

Function

This method executes the parameterized SQL statements (that is, placeholders instead of SQL literals). The psycopg2 module supports placeholders marked with **%s**.

Prototype

```
curosr.execute(query,vars_list)
```

Parameters

Table 5-86 curosr.execute parameters

Keyword	Description
query	SQL statement to be executed.
vars_list	Variable list, which matches the **%s** placeholder in the query.

Return Value

None

Examples

For details, see **Example: Common Operations**.

5.6.5.4 curosr.executemany(query,vars_list)

Function

This method executes an SQL command against all parameter sequences or mappings found in the sequence SQL.

Prototype

```
curosr.executemany(query,vars_list)
```

Parameter

Table 5-87 curosr.executemany parameters

Keyword	Description
query	SQL statement that you want to execute.
vars_list	Variable list, which matches the **%s** placeholder in the query.

Return Value

None

Examples

For details, see **Example: Common Operations**.

5.6.5.5 connection.commit()

Description

This method commits the currently pending transaction to the database.

CAUTION

By default, Psycopg opens a transaction before executing the first command. If **commit()** is not called, the effect of any data operation will be lost.

Prototype

```
connection.commit()
```

Parameter

None

Return Value

None

Examples

For details, see **Example: Common Operations**.

5.6.5.6 connection.rollback()

Function

This method rolls back the current pending transaction.

CAUTION

If you close the connection using **close()** but do not commit the change using **commit()**, an implicit rollback will be performed.

Prototype

```
connection.rollback()
```

Parameter

None

Return Value

None

Examples

For details, see **Example: Common Operations**.

5.6.5.7 cursor.fetchone()

Function

This method extracts the next row of the query result set and returns a tuple.

Prototype

```
cursor.fetchone()
```


Parameter

None

Return Value

A single tuple is the first result in the result set. If no more data is available, **None** is returned.

Examples

For details, see **Example: Common Operations**.

5.6.5.8 cursor.fetchall()

Function

This method obtains all the (remaining) rows of the query result and returns them as a list of tuples.

Prototype

```
cursor.fetchall()
```

Parameter

None

Return Value

Tuple list, which contains all results of the result set. An empty list is returned when no rows are available.

Examples

For details, see **Example: Common Operations**.

5.6.5.9 cursor.close()

Function

This method closes the cursor of the current connection.

Prototype

```
cursor.close()
```

Parameter

None

Return Value

None

Examples

For details, see **Example: Common Operations**.

5.6.5.10 connection.close()

Function

This method closes the database connection.

> ⚠ CAUTION
>
> This method closes the database connection and does not automatically call **commit()**. If you just close the database connection without calling **commit()** first, changes will be lost.

Prototype

```
connection.close()
```

Parameter

None

Return Value

None

Examples

For details, see **Example: Common Operations**.

5.7 Development Based on the Go Driver

5.7.1 Setting Up the Go Driver Environment

Go Driver Package

The package is obtained from the release package. The package name is **GaussDB-Kernel**_*Database version number*_*OS version number*_**64bit_Go.tar.gz**. Decompress the package to obtain the Go driver source code package.

> NOTICE
>
> The Go driver package provided by the database depends on Go 1.13 or later.

Environment Class

- **Configure the Go environment.**
- You need to configure the following parameters in the environment variables:
  - **GO111MODULE**: Set **GO111MODULE** to **on** when installing the Go driver by importing a file online. If you do not want to reconstruct the **Go mod** project, set **GO111MODULE** to **off** and manually download the dependency package. The dependency package must be at the same level as the driver root directory and service code.
  - **GOPROXY**: When importing data online, you need to configure the path that contains the Go driver package.
  - You can configure other Go environment variables based on your scenario parameters.

  Run the **go env** command to view the Go environment variable configuration result and check whether the Go version is 1.13 or later.
- **Install the Go driver.**
  - Download the Go driver package to the local host. The Go driver repository address is **https://open.codehub.huawei.com/OpenSourceCenter/openGauss-connector-go-pq/**.
  - Go to the root path of the Go driver code and run the **go mod tidy** command to download related dependencies. You need to configure **GOPATH=***${Path for storing the Go driver dependency package}* in the environment variables.
  - If the dependencies have been downloaded to the local host, you can add a line "Replace the Go driver package with the local Go driver package address through replace" to **go.mod**, indicating that all import Go driver packages in the code are stored in the local path and the dependencies are not downloaded from the proxy.

CAUTION

When you run the **go mod tidy** command to download dependencies, some of them may be downloaded as an earlier version. If the earlier version has vulnerabilities, you can change the dependency version in the **go.mod** file and update the dependency to the version after the vulnerability is fixed to avoid risks.

Driver Class

When creating a database connection, you need to enter the database driver name **gaussdb**.

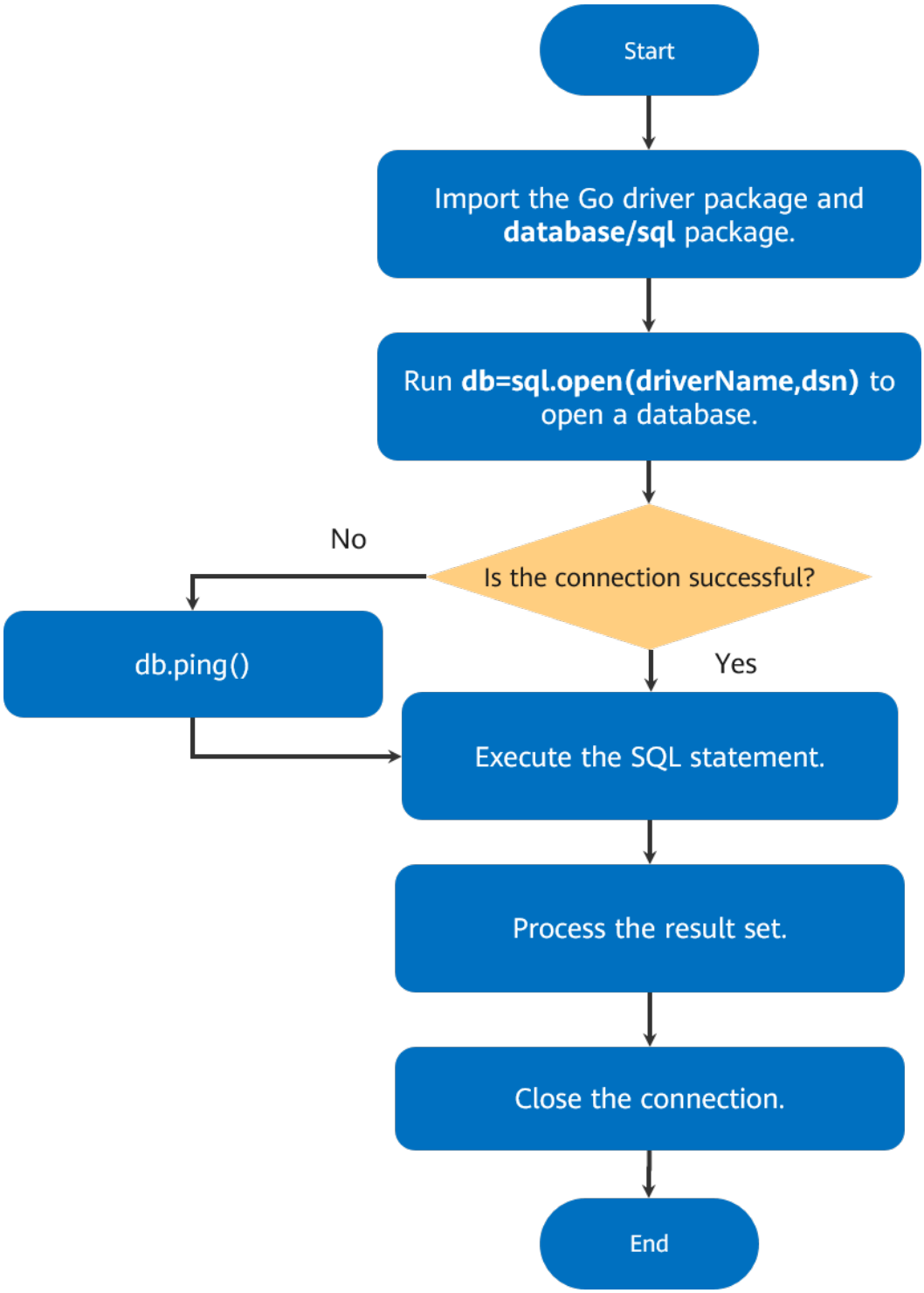
CAUTION

For details about the Go driver, see **Go Driver Compatibility**.

5.7.2 Development Process

The Go driver of the database complies with the rule of the Go language third-party library. You only need to import the driver to the application program and save the driver code in the directory specified by *GOPATH*.

Figure 5-5 Application development process based on Go



According to **Figure 5-5**, the Go driver application development process depends on the native SQL package of Go driver and the driver developed by GaussDB. The SQL package provides standard interfaces that are used for the implementation of GaussDB for users.

5.7.3 Connecting to a Database

When you call the standard SQL API **open** of the Go language to create a database connection, a connection object is returned to transfer the driver name and description string.

Function Prototype

The Go driver provides the following method to generate a database connection object:

```
func Open(driverName, dataSourceName string) (*DB, error)
```

Parameter description:

- **driverName** indicates the driver name. The database driver name is **gaussdb**.
- **dataSourceName** indicates the data source to be connected. The value can be in DSN or URL format.
  - DSN format: key1 = value1 key2 = value2.... Different groups of keywords are separated by space. The space on the left and right of the equal sign (=) is optional.
  - URL format: driverName://[userspec@][hostspec][/dbname][?paramspec]

    driverName indicates the driver name. The database driver name is **gaussdb**.

    userspec indicates user[:password]. When a URL is used for connection, the password cannot contain separators in the URL string. If the password contains separators, the DSN format is recommended.

    hostspec indicates [host][:port][, ...].

    dbname indicates the database name. Note: The initial user cannot be used for remote login. **paramspec** indicates name=value[&...].

NOTICE

- In the DSN format, if there are multiple IP addresses:
  - When the value of **num(ip)** is the same as that of **num(port)**, the IP address matches the port number.
  - When the value of **num(ip)** is greater than that of **num(port)**, the IP address that cannot match the port number matches the first port number. For example, the matching condition of **host = ip1, ip2, ip3 port = port1, port2** is **ip1:port1, ip2:port2, ip3:port1**.
  - If the value of **num(ip)** is smaller than that of **num(port)**, the extra port numbers are discarded. For example, the matching condition of **host = ip1, ip2, ip3 port = port1, port2, port3, port4** is **ip1:port1, ip2:port2, ip3:port3**.
- In the URL format, if there are multiple IP addresses:
  - In the URL, *ip*:*port* must appear in pairs, that is, the value of **num(ip)** is the same as that of **num(port)**. Use commas (,) to separate multiple pairs. Example: **gaussdb://user:password@ip1:port1, ip2:port2, ip3:port3/gaussdb**.
  - The URL contains only multiple IP addresses. The port number is specified by the environment variable or uses the default value **5432**. For example, in the case of **gaussdb://user:password@ip1, ip2, ip3/gaussdb**, ,if the environment variable *PGPORT* is set to **"port1, port2"**, the mapping is **ip1:port1, ip2:port2, ip3:port1**. If the environment variable is not set, the mapping is **ip1:5432,ip2:5432,ip3:5432**.

Parameters

Table 5-88 Database connection parameters

Parameter	Description
host	IP address of the host server, which can also be specified by the environment variable *${PGHOST}*
port	Port number of the host server, which can also be specified by the environment variable *${PGPORT}*
dbname	Database name, which can also be specified by the environment variable *${PGDATABASE}*
user	Username to be connected, which can also be specified by the environment variable *${PGUSER}*
password	Password of the user to be connected
connect_timeout	Timeout interval for connecting to the server, which can also be specified by the environment variable *${PGCONNECT_TIMEOUT}*

Parameter	Description
sslmode	SSL encryption mode, which can also be specified by the environment variable *${PGSSLMODE}* Value range: • **disable**: SSL connection is disabled. • **allow**: If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified. • **prefer**: If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified. • **require**: SSL connection is required and data is encrypted. However, authenticity of the database server will not be verified. • **verify-ca**: SSL connection is required, and whether the server certificate is issued by a trusted CA is verified. • **verify-full**: SSL connection is required, and whether the server certificate is issued by a trusted CA and whether the host name of the server is the same as that in the certificate are verified.
sslkey	Key location of the client certificate. If SSL connection is required and this parameter is not specified, you can set the environment variable *${PGSSLKEY}* to specify the location.
sslcert	File name of the client SSL certificate, which can also be specified by the environment variable *${PGSSLCERT}*
sslrootcert	Name of the file that contains the SSL CA certificate, which can also be specified by the environment variable *${PGSSLROOTCERT}*
sslcrl	File name of the SSL CRL. If a certificate listed in this file exists, the server certificate authentication will be rejected and the connection will fail. The value can also be specified by the environment variable *${PGSSLCRL}*.
sslpassword	Passphrase used to decrypt a key into plaintext. If this parameter is specified, the SSL key is an encrypted file. Currently, the SSL key supports DES encryption and AES encryption. **NOTE** The DES encryption algorithm has lower security and poses security risks. Therefore, you are advised to use a more secure encryption algorithm.
disable_prepared_binary_result	The value of this parameter is a string. If it is set to **yes**, the connection should not use the binary format when the query results are received from prepared statements. This parameter is used only for debugging. Value range: **yes** and **no**.

Parameter	Description
binary_parameters	Specifies whether **[]byte** is always sent in binary format. The value is a string. Value range: **yes** and **no**. If this parameter is set to **yes**, you are advised to bind parameters based on **[]byte** to reduce internal type conversion.
target_session_attrs	Connection type of the database, which can also be specified by the environment variable *${PGTARGETSESSIONATTRS}*. This parameter is used to identify the primary and standby nodes. There are six value options, namely, **any**, **master**, **slave**, **preferSlave**, **read-write**, and **read-only**. The default value is **any**. ● **any**: attempts to connect to any DN in the URL connection string. ● **master**: attempts to connect to a primary node in the URL connection string. If the primary node cannot be found, an exception is thrown. ● **slave**: attempts to connect to a standby node in the URL connection string. If the standby node cannot be found, an exception is thrown. ● **preferSlave**: attempts to connect to a standby DN (if available) in the URL connection string. Otherwise, it connects to the primary DN. ● **read-write**: specifies that only the primary node can be connected. ● **read-only**: specifies that only the standby node can be connected.
loggerLevel	Log level, which is used to print debugging information. The value can also be specified by the environment variable *${PGLOGGERLEVEL}*. The value can be **trace**, **debug**, **info**, **warn**, **error**, or **none**, in descending order of priority.
application_name	Name of the Go driver that is being connected. The default value is **go-driver**. You are advised not to configure this parameter.
RuntimeParams	Value of the GUC parameter of the set type that is run by default when a session is connected, for example, **search_path**, **application_name**, and **timezone**. For details about the parameters, see the default settings of the client connection. You can run the **SHOW** command to check whether the parameters are set successfully.
enable_ce	Encrypted database function. **enable_ce=1** indicates that the Go driver supports the basic capability of encrypted equality query. **enable_ce=3** indicates that the Go driver supports the fully-encrypted mode with software and hardware integrated. If this parameter is not set or set to another value, the encrypted database is disabled.

Parameter	Description
key_info	This parameter is used together with **enable_ce** to set parameters for accessing an external key manager in an encrypted database.
auto_sendtoken	Automatic key transmission function. **auto_sendtoken=yes** indicates that the key transmission is automatically triggered when the connection of the database instance connection pool is initialized. If this parameter is set to **no** or left empty, this mode is disabled. Note that this function cannot ensure real-time update of key information. In addition, this function can be enabled only when **enable_ce** is set to **3**.
tcp_syn_retries	Number of TCP connection retry times. If this parameter is not set, the default number of retry times on the device is used. You are advised to set this parameter to a value less than the value of **connect_timeout**.
socketTimeout	Read/Write timeout threshold. If the time of executing service statements or reading data streams from the network exceeds the threshold (that is, when the statement execution time exceeds the specified threshold and no data is returned), the connection is interrupted. **NOTE** This parameter specifies the maximum execution time of a single SQL statement. If the execution time of a single SQL statement exceeds the value of this parameter, the statement is interrupted and reconnected. You are advised to set this parameter based on service characteristics. If this parameter is not set, the default value **0** is used, indicating that the execution of SQL statement does not time out.

5.7.4 Connecting to a Database (Using SSL)

The Go driver supports SSL connections to the database. After the SSL mode is enabled, if the Go driver connects to the database server in SSL mode, the Go driver uses the standard TLS 1.3 protocol by default, and the TLS version must be 1.2 or later. This section describes how applications configure the client in SSL mode through the Go driver. For details about how to configure the server, contact the administrator. To use the method described in this section, you must have the server certificate, client certificate, and private key files. For details on how to obtain these files, see related documents and commands of OpenSSL.

NOTE

In SSL-based certificate authentication mode, you do not need to specify the user password in the connection string.

Configuring the Client

Upload the **client.key**, **client.crt**, and **cacert.pem** files generated during server configuration to the client. For details about server configuration, contact the administrator.

Example 1:

```
package main

// Set the dependency package based on the dependency package path in the environment.
import (
 "database/sql"
 "fmt"
 _ "gitee.com/opengauss/openGauss-connector-go-pq"
 "log"
)
// Mutual authentication is used as an example. In this example, the username and password are stored in environment variables. Before running this example, set environment variables in the local environment (set the environment variable name based on the actual situation).
func main() {
    hostip := os.Getenv("GOHOSTIP")   // GOHOSTIP indicates the IP address written into the environment variable.
    port := os.Getenv("GOPORT")       // GOPORT indicates the port number written into the environment variable.
    usrname := os.Getenv("GOUSRNAME") // GOUSRNAME indicates the username written into the environment variable.
    passwd := os.Getenv("GOPASSWD")   // GOPASSWD indicates the user password written into the environment variable.
    sslpasswd := os.Getenv("GOSSLPASSWD") // GOSSLPASSWD indicates the passphrase written into the environment variable.
  dsnStr := "host=" + hostip + " port=" + port + " user=" + usrname + " password=" + passwd + " dbname=gaussdb sslcert=certs/client.crt sslkey=certs/client.key sslpassword=" + sslpasswd
 parameters := []string{
  " sslmode=require",
  " sslmode=verify-ca sslrootcert=certs/cacert.pem",
 }

 for _, param := range parameters {
  db, err := sql.Open("gaussdb", dsnStr+param)
  if err != nil {
   log.Fatal(err)
  }

  var f1 int
  err = db.QueryRow("select 1").Scan(&f1)
  if err != nil {
   log.Fatal(err)
  } else {
   fmt.Printf("RESULT: select 1: %d\n", f1)
  }

  db.Close()
 }
}
```

Example 2:

```
package main

// Set the dependency package based on the dependency package path in the environment.
import (
 "database/sql"
 _ "gitee.com/opengauss/openGauss-connector-go-pq"
    "log"
    "strings"
)
// For example, verify sslpassword. In this example, the username and password are stored in environment variables. Before running this example, set environment variables in the local environment (set the environment variable name based on the actual situation).
func main() {
    hostip := os.Getenv("GOHOSTIP")   // GOHOSTIP indicates the IP address written into the environment variable.
    port := os.Getenv("GOPORT")       // GOPORT indicates the port number written into the environment variable.
    usrname := os.Getenv("GOUSRNAME") // GOUSRNAME indicates the username written into the
```



```
environment variable.
    passwd := os.Getenv("GOPASSWD")    // GOPASSWD indicates the user password written into the
environment variable.
    dsnStr := "host=" + hostip + " port=" + port + " user=" + usrname + " password=" + passwd + "
dbname=gaussdb"
    sslpasswd := os.Getenv("GOSSLPASSWD") // GOSSLPASSWD indicates the passphrase written into the
environment variable.
   connStrs := []string{
      " sslmode=verify-ca sslcert=certs/client_rsa.crt sslkey=certs/client_rsa.key sslpassword=" + sslpasswd +
" sslrootcert=certs/cacert_rsa.pem",
      " sslmode=verify-ca sslcert=certs/client_ecdsa.crt sslkey=certs/client_ecdsa.key sslpassword=" +
sslpasswd + " sslrootcert=certs/cacert_ecdsa.pem",
   }
   for _, connStr := range connStrs {
      db, err := sql.Open("gaussdb", dsnStr+connStr)
      if err != nil {
         log.Fatal(err)
      }
      var f1 int
      err = db.QueryRow("select 1").Scan(&f1)
      if err != nil {
         if !strings.HasPrefix(err.Error(), "connect failed.") {
            log.Fatal(err)
         }
      }
      db.Close()
   }
}
```

5.7.5 Go APIs

5.7.5.1 sql.Open

The following table describes sql.Open.

Method	Description	Return Value
Open(driverName, dataSourceName string)	Opens a database based on a specified database driver and the dedicated data source of the driver.	*DB and error

For details about the **driverName** and **dataSourceName** parameters, see **Connecting to a Database**.

5.7.5.2 type DB

The following table describes type DB.

Method	Description	Return Value
(db *DB)Begin()	Starts a transaction. The isolation level of the transaction is determined by the driver.	*Tx and error

Method	Description	Return Value
(db *DB)BeginTx(ctx context.Context, opts *TxOptions)	Starts a transaction with a specified transaction isolation level. A specified context is used until the transaction is committed or rolled back. If the context is canceled, the SQL package rolls back the transaction.	*Tx and error
(db *DB)Close()	Closes the database and releases all the opened resources.	error
(db *DB)Exec(query string, args ...interface{})	Performs an operation that does not return rows of data.	Result and error
(db *DB)ExecContext(ctx context.Context, query string, args ...interface{})	Performs an operation that does not return rows of data in a specified context.	Result and error
(db *DB)Ping()	Checks whether the database connection is still valid and establishes a connection if necessary.	error
(db *DB)PingContext(ctx context.Context)	Checks whether the database connection is still valid in a specified context and establishes a connection if necessary.	error
(db *DB)Prepare(query string)	Creates a prepared statement for subsequent queries or executions.	*Stmt and error
(db *DB)PrepareContext(ctx context.Context, query string)	Creates a prepared statement for subsequent queries or executions in a specified context.	*Stmt and error
(db *DB)Query(query string, args ...interface{})	Executes a query and returns multiple rows of data.	*Rows and error
(db *DB)QueryContext(ctx context.Context, query string, args ...interface{})	Executes a query and returns multiple rows of data in a specified context.	*Rows and error
(db *DB)QueryRow(query string, args ...interface{})	Executes a query that returns only one row of data.	*Row

Method	Description	Return Value
(db *DB)QueryRowContext(ctx context.Context, query string, args ...interface{})	Executes a query that returns only one row of data in a specified context.	*Row

Parameters

Parameter	Description
ctx	Specified context
query	Executed SQL statement
args	Parameter that needs to be bound to the executed SQL statement. Binding by location and binding by name are supported. For details, see **Examples**.
opts	Transaction isolation level and transaction access mode. The transaction isolation level (**opts.Isolation**) supports sql.LevelReadUncommitted, sql.LevelReadCommitted, sql.LevelRepeatableRead, and sql.LevelSerializable. The transaction access mode (opts.ReadOnly) can be true (read only) or false (read write).

NOTICE

1. The Query(), QueryContext(), QueryRow(), and QueryRowContext() APIs are usually used in query statements, such as SELECT. The Exec() API is used for executing operation statements. If query APIs are used to execute non-query statements, the execution result may be unexpected. Therefore, you are advised not to use the query APIs to execute non-query statements, such as UPDATE and INSERT.
2. The result of executing a query statement using a query API needs to be obtained through the Next() API in **type Rows**. If the result is not obtained through the Next() API, unexpected errors may occur.

Examples

```
// In this example, the username and password are stored in environment variables. Before running this example, set environment variables in the local environment (set the environment variable names based on the actual situation).
package main
// Set the dependency package based on the dependency package path in the environment.
import (
 "database/sql"
 _ "gitee.com/opengauss/openGauss-connector-go-pq"
 "log"
)
```



```
func main() {
	hostip := os.Getenv("GOHOSTIP")   // GOHOSTIP indicates the IP address written into the
environment variable.
	port := os.Getenv("GOPORT")       // GOPORT indicates the port number written into the environment
variable.
	usrname := os.Getenv("GOUSRNAME") // GOUSRNAME indicates the username written into the
environment variable.
	passwd := os.Getenv("GOPASSWD")   // GOPASSWDW indicates the user password written into the
environment variable.
	str := "host=" + hostip + " port=" + port + " user=" + usrname + " password=" + passwd + "
dbname=gaussdb sslmode=disable"
 db, err := sql.Open("gaussdb", str)
 if err != nil {
  log.Fatal(err)
 }
 defer db.Close()

 err = db.Ping()
 if err != nil {
  log.Fatal(err)
 }

 _, err = db.Exec("create table test_bound(id int, name text)")

    // Binding by location
    _, err = db.Exec("insert into test_bound(id, name) values(:1, :2)", 1, "Zhang San")
 if err != nil {
  log.Fatal(err)
 }

    // Binding by name
    _, err = db.Exec("insert into test_bound(id, name) values(:id, :name)", sql.Named("id", 1),
sql.Named("name", "Zhang San"))
 if err != nil {
  log.Fatal(err)
 }
}
```

5.7.5.3 type Stmt

The following table describes type Stmt.

Method	Description	Return Value
(s *Stmt)Close()	Closes a specified prepared statement.	error

(s *Stmt)Exec(args ...interface{})	Executes a prepared statement with specified parameters and returns a **Result** value. The Prepare-Bind-Execute (PBE) feature is supported. PBE is a method of sending and executing queries. The CN can receive PBE packets through complex query protocols to execute statements.	Result and error

	NOTE 1. The placeholder of a precompiled statement can be a dollar sign ($) or a question mark (?). 2. The number of placeholders in a precompiled statement is determined by the database. When the number of table columns exceeds the database limit or does not match the number of current table columns, the server returns an error. 3. Batch PBE processing supports addition, deletion, and modification. During the batch operation, the maximum length of a U packet is limited to 1 GB minus 1 byte, that is, 0x3fffffff bytes. If the length exceeds the limit, "bind message length XXX too long. This can be caused by very large or incorrect ength specifications on InputStream parameters" will be reported. 4. When a record is inserted, the performance of PBE deteriorates greatly compared with that of a single-query statement (conn.simpleExec). Therefore, you are advised to use single-query statements, rather than the PBE statement. 5. After the underlying error processing of the driver is reconstructed, the PBE performance decreases by less than 5%.	

(s *Stmt)ExecContext(ctx context.Context, args ...interface{})	Executes a prepared statement with specified parameters in a specified context and returns a **Result** value.	Result and error
(s *Stmt)Query(args ...interface{})	Executes a prepared statement with specified parameters and returns ***Rows** as the query result.	*Rows and error
(s *Stmt)QueryContext(ctx context.Context, args ...interface{})	Executes a prepared statement with specified parameters in a specified context and returns ***Rows** as the query result.	*Rows and error
(s *Stmt)QueryRow(args ...interface{})	Executes a prepared statement with specified parameters and returns ***Row** as the result.	*Row
(s *Stmt)QueryRowContext (ctx context.Context, args ...interface{})	Executes a prepared statement with specified parameters in a specified context and returns ***Row** as the result.	*Row

Parameters

Parameter	**Description**
ctx	Specified context
query	Executed SQL statement
args	Parameter that needs to be bound to the executed SQL statement. Binding by location and binding by name are supported. For details, see **Examples** in section "type DB."

NOTICE

1. The Query(), QueryContext(), QueryRow(), and QueryRowContext() APIs are usually used in query statements, such as SELECT. The Exec() API is used for executing operation statements. If query APIs are used to execute non-query statements, the execution result may be unexpected. Therefore, you are advised not to use the query APIs to execute non-query statements, such as UPDATE and INSERT.
2. The result of executing a query statement using a query API needs to be obtained through the Next() API in **type Rows**. If the result is not obtained through the Next() API, unexpected errors may occur.

5.7.5.4 type Tx

The following table describes type Tx.

Method	Description	Return Value
(tx *Tx)Commit()	Commits a transaction.	error
(tx *Tx)Exec(query string, args ...interface{})	Performs an operation that does not return rows of data.	Result and error
(tx *Tx)ExecContext(ctx context.Context, query string, args ...interface{})	Performs an operation that does not return rows of data in a specified context.	Result and error
(tx *Tx)Prepare(query string)	Creates a prepared statement for subsequent queries or executions. The returned statement is executed within a transaction and cannot be used when the transaction is committed or rolled back.	*Stmt and error

(tx *Tx)PrepareContext(ctx context.Context, query string)	Creates a prepared statement for subsequent queries or executions. The returned statement is executed within a transaction and cannot be used when the transaction is committed or rolled back. The specified context will be used in the preparation phase, not in the transaction execution phase. The statement returned by this method will be executed in the transaction context.	*Stmt and error
(tx *Tx)Query(query string, args ...interface{})	Executes a query that returns rows of data.	*Rows and error
(tx *Tx)QueryContext(ctx context.Context, query string, args ...interface{})	Executes a query that returns rows of data in a specified context.	*Rows and error
(tx *Tx)QueryRow(query string, args ...interface{})	Executes a query that returns only one row of data.	*Row
(tx *Tx)QueryRowContext(ctx context.Context, query string, args ...interface{})	Executes a query that returns only one row of data in a specified context.	*Row
(tx *Tx) Rollback()	Rolls back a transaction.	error
(tx *Tx)Stmt(stmt *Stmt)	Returns a transaction-specific prepared statement for an existing statement. Example: `str, err := db.Prepare("insert into t1 values(:1, :2)")` `tx, err := db.Begin()` `res, err := tx.Stmt(str).Exec(1, "aaa")`	*Stmt
(tx *Tx)StmtContext(ctx context.Context, stmt *Stmt)	Returns a transaction-specific prepared statement for an existing statement in a specified context.	*Stmt

Parameters

Parameter	Description
ctx	Specified context
query	Executed SQL statement
args	Parameter that needs to be bound to the executed SQL statement. Binding by location and binding by name are supported. For details, see **Examples** in section "type DB."
stmt	Existing prepared statement, which is generally the prepared statement returned by the PREPARE statement

NOTICE

1. The Query(), QueryContext(), QueryRow(), and QueryRowContext() APIs are usually used in query statements, such as SELECT. The Exec() API is used for executing operation statements. If query APIs are used to execute non-query statements, the execution result may be unexpected. Therefore, you are advised not to use the query APIs to execute non-query statements, such as UPDATE and INSERT.
2. The result of executing a query statement using a query API needs to be obtained through the Next() API in **type Rows**. If the result is not obtained through the Next() API, unexpected errors may occur.

5.7.5.5 type Rows

The following table describes type Rows.

Method	Description	Return Value
(rs *Rows)Close()	Closes **Rows** to stop the iteration of the data set.	error
(rs *Rows)ColumnTypes()	Returns column information.	[]*ColumnType and error
(rs *Rows)Columns()	Returns the name of each column.	[]string and error
(rs *Rows)Err()	Returns any errors that occur during iteration.	error

(rs *Rows)Next()	Prepares the next data row to be read with the Scan method. If there is an additional result set, **true** is returned. Otherwise, **false** is returned.	boolean
(rs *Rows)Scan(dest ...interface{})	Copies the columns of the current iterated row of data to the value specified by **dest**.	error
(rs *Rows)NextResultSet()	Specifies whether there is an additional result set.	boolean

Parameters

Parameter	**Description**
dest	Queries the value to which the column needs to be copied.

5.7.5.6 type Row

The following table describes type Row.

Method	**Description**	**Return Value**
(r *Row)Scan(dest ...interface{})	Copies the columns in the current row of data to the value specified by **dest**.	error
(r *Row)Err()	Returns errors that occur during execution.	error

Parameter Description

Parameter	Description
dest	The column to be queried needs to be copied to the value specified by this parameter.

5.7.5.7 type ColumnType

The following table describes type ColumnType.

Method	**Description**	**Return Value**

(ci *ColumnType)DatabaseTypeName()	Returns the name of the column-type database system. If an empty string is returned, driver-type names are not supported.	error
(ci *ColumnType)DecimalSize()	Returns the scale and precision of the decimal type. If the value of **ok** is **false**, the specified type is unavailable or not supported.	precision, scale int64, ok boolean
(ci *ColumnType)Length()	Returns the length of the data column type. If the value of **ok** is **false**, the specified type does not have a length.	length int64, ok boolean
(ci *ColumnType)ScanType()	Returns a Go type that can be used for scanning by using Rows.Scan.	reflect.Type
(ci *ColumnType)Name()	Returns the name of a data column.	string

5.7.5.8 type Result

The following table describes type Result.

Method	Description	Return Value
(res Result)RowsAffected()	Returns the number of rows affected by the INSERT, DELETE, UPDATE, SELECT, MOVE, FETCH, and COPY operations.	int64 and error

5.8 ECPG-based Development

Embedded SQL C Preprocessor (ECPG) for GaussDB Kernel is an embedded SQL preprocessor for C programs. An embedded SQL program consists of code written in an ordinary programming language, in this case C, mixed with SQL commands in specially marked sections. To build the program, the source code (*.pgc) is first passed through the embedded SQL preprocessor, which converts it to an ordinary C program (*.c), and afterwards it can be processed by a C compiler. Converted ECPG applications call functions in the libpq library through the embedded SQL library (ecpglib), and communicate with the GaussDB Kernel server using the normal frontend-backend protocol.

Programs written for the embedded SQL interface are normal C programs with special code inserted to perform database-related actions. This special code always has the form:

```
EXEC SQL ...;
```

These statements syntactically take the place of a C statement. Depending on the particular statement, they can appear at the global level or within a function. Embedded SQL statements follow the case-sensitivity rules of normal SQL code, and allow nested C code-style comments (part of the SQL standard). However, the C part of the program follows the standards of the C program and does not support nested comments.

5.8.1 Development Process

Figure 5-6 ECPG-based development process

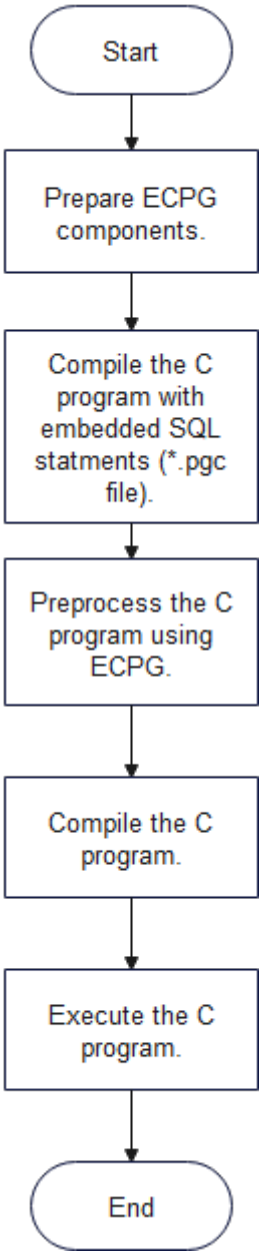
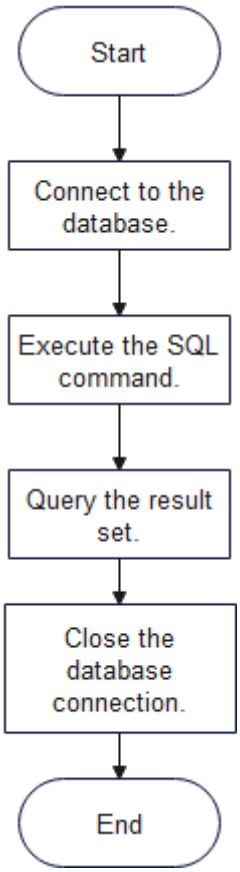


Figure 5-7 Embedded SQL-C program development process



5.8.2 ECPG Components

- Platforms supported by ECPG

  Table 5-89 Platforms supported by ECPG

OS	Platform
EulerOS V2.0SP5	x86_64
EulerOS V2.0SP9	Arm64
Kylin V10	x86_64
Kylin V10	Arm64

- ecpg components
  - ecpg: an executable binary file used to preprocess C programs with embedded SQL statements.
  - libecpg: dynamic library provided by ecpg to implement connections, SQL statements, and transactions, including **libecpg.so**, **libecpg.so.6**, and **libecpg.so.6.4**. It is referenced by the **-lecpg** parameter during C program compilation and execution.
  - libpgtypes: dynamic library provided by the ECPG for operating data of the numeric, date, timestamp, and interval types, including **libpgtypes.so**,

libecpg.so.6, and **libecpg.so.6.4**. The library is referenced by the **-lpgtypes** parameter during C program compilation and execution.

- Paths for obtaining ECPG components
  - ECPG binary file: *$GAUSSHOME*/**bin**
  - Dynamic library on which the ECPG depends: *$GAUSSHOME*/**lib**
  - Header file required by ECPG: *$GAUSSHOME*/**include/ecpg**

5.8.3 ECPG Preprocessing and Compiling

Prepare C programs with embedded SQL statements with the extension .pgc. ECPG converts them into C programs that can be compiled by the C compiler.

The generated C program is compiled into an executable file by the compiler. The executable file is run to enable the client program to access the database. For details, see **Examples**.

- ecpg preprocessing and C compilation process
  a. Preprocessing: ecpg -I $GAUSSHOME/include -o test.c test.pgc

     To execute ECPG preprocessing, run the following command:

     ```
     ecpg [OPTION]...
     ```

     The options are as follows:

     - **-o OUTFILE**: writes the result to OUTFILE, which is a C file.
     - **-I DIRECTORY**: path of the header file.
     - **-c**: automatically generates a C file.
     - **--version**: checks the current ECPG version.

  b. Compilation: gcc -I $GAUSSHOME/include/ecpg -I $GAUSSHOME/include -I $GAUSSHOME/include/gaussdb/server/ -L $GAUSSHOME/lib -lecpg -lrt -lpq -lpgtypes -lpthread test_ecpg.c -o test_ecpg

  c. Execution: ./test

NOTICE

- ECPG is a compilation preprocessing tool. If an error message is displayed indicating that the related header file or function implementation cannot be found during preprocessing or compilation, you can specify the header file or link the dynamic library as required.
- ECPG requires compilation preprocessing tools such as GCC and ld. You are advised to use GCC 7.3.0.
- Other dynamic libraries and header files on which the ECPG depends are usually stored in *$GAUSSHOME*/**include/libpq** and *$GAUSSHOME*/**include**.
- Common dynamic library dependencies during compilation include: -lpq, -lpq_ce, and -lpthread. If the libpq communications library is required during development, connect to -lpq and -lpq_ce. If the multi-thread connection is required during development, connect to -lpthread.

5.8.4 Managing Database Connections

This section describes how to establish and switch a database connection.

5.8.4.1 Connecting to a Database

You can run the following command to connect to the database:

```
EXEC SQL CONNECT TO target [AS connection-name] [USER user-name];
```

The target can be declared using the following methods. The italic part is a variable. Change it based on the actual situation.

- *dbname*[@*hostname*][:*port*]
- tcp:gaussdb://*hostname*[:*port*][/*dbname*][?*options*]
- unix:gaussdb://*hostname*[:*port*][/*dbname*][?*options*]
- An SQL string containing one of the above forms

There are also different ways to specify the username:

- *username*/*password*
- *username* SQLIDENTIFIED BY *password*
- *username* USING *password*

As mentioned above, the **username** and **password** parameters can be an SQL identifier, an SQL string, or a reference to a character variable.

connection_name indicates the connection name. If a program uses only one connection, you can omit it. The most recently opened connection becomes the current connection.

The following is an example:

```
#include <stdlib.h>
EXEC SQL CONNECT TO mydb@sql.mydomain.com;

EXEC SQL CONNECT TO unix:gaussdb://sql.mydomain.com/mydb AS myconnection USER username;

EXEC SQL BEGIN DECLARE SECTION;
/* The values of target, user, and passwd must be read from environment variables or configuration files. Environment variables need to be configured as required. If no environment variable is used, a character string can be directly assigned. */
const char *target = getenv("EXAMPLE_TARGET_ENV");
const char *user = getenv("EXAMPLE_USERNAME_ENV");
const char *passwd = getenv("EXAMPLE_PASSWD_ENV");
EXEC SQL END DECLARE SECTION;
...
EXEC SQL CONNECT TO :target USER :user USING :passwd;
/* or EXEC SQL CONNECT TO :target USER :user/:passwd; */
```

For details about the complete usage example, see the example of using the connection syntax in **CONNECT**.

NOTE

- In the last form, character variables are referenced. For details about how to reference C variables in SQL statements, see **Host Variables**.
- The format of the connection target is not described in the SQL standard. Therefore, to develop a portable application, you can use the method in the last example to encapsulate the connection target string into a variable.
- For details about the ecpg compatibility, see **ecpg Compatibility**.

NOTICE

- If **ip-port** is specified in the connection statement, username and password must be specified. This rule is determined by the GaussDB Kernel communication authentication. If **ip-port** is not specified, the local *$PGPORT* (UDS) is used for communication.
- If the SSL protocol is used for connection, run the **tcp:gaussdb://** *hostname*[**:***port*][**/***dbname*][**?***options*] command and set **sslmode** to **disable \require** in **options**.

5.8.4.2 Managing Connections

SQL statements in embedded SQL programs are by default executed on the current connection, that is, the most recently opened one. If an application needs to manage multiple connections, use either of the following methods:

- Method 1: Explicitly select a connection for each SQL statement.
  ```
  EXEC SQL AT connection-name SELECT ...;
  ```
  This method is particularly suitable if the application needs to use several connections in mixed order.

  If the application uses multiple threads of execution, they cannot share a connection concurrently. You must either explicitly control access to the connection (using mutexes) or use a connection for each thread.
- Method 2: Execute a statement to switch the connection.
  ```
  EXEC SQL SET CONNECTION connection-name;
  ```
  This method is particularly suitable if many statements are executed on the same connection.

The following is an example of managing connections.

```
#include <stdio.h>
EXEC SQL BEGIN DECLARE SECTION;
   char dbname[1024];
EXEC SQL END DECLARE SECTION;

int main()
{
    EXEC SQL CONNECT TO testdb1 AS con1 USER testuser;
    EXEC SQL CONNECT TO testdb2 AS con2 USER testuser;
    EXEC SQL CONNECT TO testdb3 AS con3 USER testuser;

    /* This query will be executed in the most recently opened database testdb3. */
    EXEC SQL SELECT current_database() INTO :dbname;
    printf("current=%s (should be testdb3)\n", dbname);

    /* Use AT to run a query in testdb2. */
    EXEC SQL AT con2 SELECT current_database() INTO :dbname;
    printf("current=%s (should be testdb2)\n", dbname);

    /* Switch to connection to testdb1. */
    EXEC SQL SET CONNECTION con1;

    EXEC SQL SELECT current_database() INTO :dbname;
    printf("current=%s (should be testdb1)\n", dbname);

    EXEC SQL DISCONNECT ALL;
    return 0;
}
```

Example output:


```
current=testdb3 (should be testdb3)
current=testdb2 (should be testdb2)
current=testdb1 (should be testdb1)
```

NOTICE

- In multi-thread mode, different threads cannot use the same connection name. The connection name of each thread must be unique.
- A connection must be established and closed in the same process or thread.

5.8.5 Running SQL Commands

The format of embedded SQL commands is EXEC SQL [Command]. In embedded SQL applications, you can run common standard SQL statements supported by GaussDB Kernel or extended SQL statements provided by ECPG. Currently, features or syntaxes such as stored procedures, packages, anonymous blocks, and flashback are not supported.

5.8.5.1 Executing SQL Statements

Step 1 Create a table.

```
EXEC SQL CREATE TABLE foo (a int, b varchar);
```

Step 2 Insert a row.

```
EXEC SQL INSERT INTO foo VALUES (5, 'abc');
```

Step 3 Delete a row.

```
EXEC SQL DELETE FROM foo WHERE a = 5;
```

Step 4 Update table data.

```
EXEC SQL UPDATE foo SET b = 'gdp' WHERE a = 7;
```

Step 5 Query data in a single row.

```
EXEC SQL SELECT a INTO :var_a FROM foo WHERE b = 'def';
```

----End

A complete example is as follows:

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int main ()
{
   ECPGdebug (1, stderr);

   EXEC SQL BEGIN DECLARE SECTION;
   int var_a;
   EXEC SQL END DECLARE SECTION;
   /* Create testdb in advance. */
   EXEC SQL CONNECT TO testdb;
   // Create a table.
   EXEC SQL CREATE TABLE foo (a int, b varchar);
      // Insert data.
   EXEC SQL INSERT INTO foo VALUES (5, 'abc');
   EXEC SQL INSERT INTO foo VALUES (6, 'def');
   EXEC SQL INSERT INTO foo VALUES (7, 'ghi');

   // Delete a row.
```



```
    EXEC SQL DELETE FROM foo WHERE a = 5;
    // Update table data.
    EXEC SQL UPDATE foo SET b = 'gdp' WHERE a = 7;
    // Query table data in a single row.
    EXEC SQL SELECT a INTO :var_a FROM foo WHERE b = 'def';
    // Print the query results.
    printf("select res is %d\n", var_a);

    EXEC SQL DISCONNECT;

    return 0;
}
```

5.8.5.2 Using Cursors

To retrieve a result set holding multiple rows, an application has to declare a cursor and fetch each row from the cursor.

Step 1 Declare a cursor.

```
EXEC SQL DECLARE c CURSOR FOR select * from tb1;;
```

Step 2 Open a cursor.

```
EXEC SQL OPEN c;
```

Step 3 Fetch a row of data from a cursor.

```
EXEC SQL FETCH 1 in c into :a, :str;
```

Step 4 Close a cursor.

```
EXEC SQL CLOSE c;
```

----End

For details about how to use cursors, see **DECLARE**. For details about the **FETCH** command, see **FETCH**.

A complete example is as follows:

```
#include <string.h>
#include <stdlib.h>

int main(void)
{
exec sql begin declare section;
    int *a = NULL;
    char *str = NULL;
exec sql end declare section;

    int count = 0;
    /* Create testdb in advance. */
    exec sql connect to testdb ;
    exec sql set autocommit to off;
    exec sql begin;
    exec sql drop table if exists tb1;
    exec sql create table tb1(id int, info text);
    exec sql insert into tb1 (id, info) select generate_series(1, 100000), 'test';
    exec sql select count(*) into :a from tb1;
    printf ("a is %d\n", *a);
    exec sql commit;

    // Define a cursor.
    exec sql declare c cursor for select * from tb1;
    // Open the cursor.
    exec sql open c;
    exec sql whenever not found do break;
    while(1) {
        // Capture data.
        exec sql fetch 1 in c into :a, :str;
```



```
        count++;
        if (count == 100000) {
            printf("Fetch res: a is %d, str is %s", *a, str);
        }
    }
    // Close the cursor.
    exec sql close c;
    exec sql set autocommit to on;
    exec sql drop table tb1;
    exec sql disconnect;

    ECPGfree_auto_mem();
    return 0;

}
```

5.8.5.3 Managing Transactions

In the default mode, statements are committed only when EXEC SQL COMMIT is issued. The embedded SQL interface also supports autocommit of transactions by executing the **EXEC SQL SET AUTOCOMMIT TO ON** statement. In autocommit mode, each command is automatically committed unless it is inside an explicit transaction block. This mode can be explicitly turned off by using **EXEC SQL SET AUTOCOMMIT TO OFF**.

Common transaction management commands are as follows:

- **EXEC SQL COMMIT**: commits an ongoing transaction.
- **EXEC SQL ROLLBACK**: rolls back an ongoing transaction.
- **EXEC SQL SET AUTOCOMMIT TO ON**: enables the autocommit mode.
- **SET AUTOCOMMIT TO OFF**: disables the autocommit mode. This is the default mode.

5.8.5.4 Prepared Statements

Prepared statements can be used when the value passed to an SQL statement is unknown at compile time or the same statement will be used multiple times.

- Statements are prepared using the **PREPARE** command. For the values that are not known yet, use the placeholder (?).
  ```
  EXEC SQL PREPARE stmt1 FROM "SELECT oid, datname FROM pg_database WHERE oid = ?";
  ```
- If a statement returns a single row, the application can call EXECUTE after PREPARE to execute the statement, supplying the actual values for the placeholders with a USING clause:
  ```
  EXEC SQL EXECUTE stmt1 INTO :dboid, :dbname USING 1;
  ```
- If a statement returns multiple rows, the application can use a cursor declared based on the prepared statement. To bind input parameters, the cursor must be opened with a USING clause:
  ```
  EXEC SQL PREPARE stmt1 FROM "SELECT oid,datname FROM pg_database WHERE oid > ?";
  EXEC SQL DECLARE foo_bar CURSOR FOR stmt1;
  /* When the end of the result set is reached, exit the while loop. */
  EXEC SQL WHENEVER NOT FOUND DO BREAK;
  EXEC SQL OPEN foo_bar USING 100;
  ...
  while (1)
  {
      EXEC SQL FETCH NEXT FROM foo_bar INTO :dboid, :dbname;
      ...
  }
  EXEC SQL CLOSE foo_bar;
  ```


- When a prepared statement is no longer needed, it should be deallocated.
  ```
  EXEC SQL DEALLOCATE PREPARE name;
  ```

5.8.5.5 Embedded SQL Commands

5.8.5.5.1 ALLOCATE DESCRIPTOR

Function

Allocates a newly named SQL descriptor area.

Syntax

```
ALLOCATE DESCRIPTOR name
```

Parameters

name

Name of an SQL descriptor, case sensitive. This can be an SQL identifier or a host variable.

Examples

```
EXEC SQL ALLOCATE DESCRIPTOR mydesc;
```

Helpful Links

DEALLOCATE DESCRIPTOR, **GET DESCRIPTOR**, **SET DESCRIPTOR**

5.8.5.5.2 CONNECT

Description

Establishes a connection between the client and the SQL server.

Syntax

```
CONNECT TO connection_target [ AS connection_name ] [ USER connection_user ]
```

Parameters

- **connection_target**

  Specifies the target server to be connected in one of the following formats:

  - [*database_name*] [@ *host*] [: *port*]: connection over TCP/IP.
  - unix:gaussdb://*host* [: *port*] / [*database_name*] [? *connection_option*]: connection over UDSs.
  - tcp:gaussdb://*host* [:*port*] / [*database_name*] [?*connection_option*]: connection over TCP/IP.
  - **SQL string constant**: one of the preceding forms.

> **CAUTION**
>
> For details about **connection_target**, see **ecpg Compatibility**.

- **connection_name**

  An optional identifier used for the connection, which can be referenced in other commands. It can be an SQL identifier or a host variable.

- **connection_user**

  Username for database connection.

  You can use *user_name*/*password*, *user_name* **SQLIDENTIFIED BY** *password*, or *user_name* **USING** *password* to specify the username and password.

  The username and password can be SQL identifiers, string constants, or host variables.

> **NOTE**
>
> In the preceding parameters, the information in italics refers to variables. Replace them based on the actual situation.

Examples

Here are several variants of specifying connection parameters:

```
EXEC SQL CONNECT TO "connectdb" AS main;
EXEC SQL CONNECT TO "connectdb" AS second;
EXEC SQL CONNECT TO 'connectdb' AS main;
EXEC SQL CONNECT TO REGRESSDB1 as main;
EXEC SQL CONNECT TO connectdb AS :id;
EXEC SQL CONNECT TO connectdb AS main USER connectuser/connectdb;
EXEC SQL CONNECT TO connectdb AS main USER connectuser USING "connectdb";
EXEC SQL CONNECT TO connectdb AS main;
EXEC SQL CONNECT TO tcp:gaussdb://localhost/connectdb USER connectuser IDENTIFIED BY connectpw;
EXEC SQL CONNECT TO tcp:gaussdb://localhost:$PORT/connectdb USER connectuser SQLIDENTIFIED BY
connectpw;
EXEC SQL CONNECT TO unix:gaussdb://localhost/connectdb USER connectuser SQLIDENTIFIED BY
"connectpw";
EXEC SQL CONNECT TO unix:gaussdb://localhost/connectdb USER connectuser USING "connectpw";
```

The following is an example of using the connection syntax:

```
#include <stdlib.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
// Define the host by defining columns such as database and password required by the connection string. The actual values are read from environment variables or configuration files. Environment variables need to be configured as required. If no environment variable is used, a character string can be directly assigned.
exec sql begin declare section;
    const int max_str_len = 200;
    char db[max_str_len] = getenv("EXAMPLE_DATABASENAME_ENV");
    char pw[max_str_len] = getenv("EXAMPLE_PASSWD_ENV");
    char new_pw[max_str_len] = getenv("EXAMPLE_NEW_PASSWD_ENV");
exec sql end declare section;

    // Print debug logs.
    ECPGdebug(1, stderr);

    // The connection statement involves the database, username, and password. The user must be created in advance and have related operation permissions.
```



```
    //Connection mode: EXEC SQL CONNECT TO [ database_name ][ @host ][ :port ] [ USER
connection_user ]
    // Case 1: Use the default local connection mode to connect to the postgres database.
    exec sql connect to postgres;
    // Case 2: Use the default local connection mode to connect to the postgres database. The connection
alias is conn1.
    exec sql connect to postgres as conn1;
    // Case 3: Use the ip+port mode (localhost indicates the local IP address listened by the database, and
$PORT indicates the listening port of the database) to connect to the connectdb database, specify the
database alias, and specify the user password.
    exec sql connect to connectdb@localhost:$PORT as conn2 user connectuser using :pw;
    // Case 4: Use the ip+port mode (127.0.0.1 indicates the local address listened by the database, and
$PORT indicates the listening port of the database) to connect to the connectdb database, specify the
database alias, and specify the user password.
    exec sql connect to connectdb@127.0.0.1:$PORT as conn3 user connectuser sqlidentified by :pw;
    // Case 5: Close the connection to the database.
    exec sql disconnect postgres;
    exec sql disconnect conn1;
    exec sql disconnect conn2;
    exec sql disconnect conn3;

    // Connection mode: EXEC SQL CONNECT TO <tcp|unix>:<gaussdb|postgresql>://host
[ :port ]/[ database_name ][ ?connection_option ]
    // Case 1: Replace the URL variables with the host variables pw and db.
    strcpy(pw, new_pw);
    strcpy(db, "tcp:postgresql://localhost/connectdb");
    exec sql connect to :db user connectuser using :pw;
    // Case 2: 127.0.0.1 indicates the IP address listened by the database, and connectdb indicates the
database.
    exec sql connect to tcp:postgresql://127.0.0.1/connectdb as conn4 user connectuser using :pw;
    // Case 3: 127.0.0.1 indicates the IP address listened by the database, connectdb indicates the database,
and connect_timeout=14 indicates the connection string configuration parameter.
    exec sql connect to tcp:gaussdb://localhost/connectdb?connect_timeout=14 as conn5 user connectuser
sqlidentified by :pw;
    // Case 4: Close all connections.
    exec sql close all;
    // Connect to the database and execute the service.
    exec sql connect to tcp:postgresql://127.0.0.1/connectdb as conn4 user connectuser using :pw;
    exec sql set autocommit = on;
    exec sql create table t1(a int);
    exec sql insert into t1 values(1),(2);
    exec sql select a from t1 where a > 1;
    exec sql drop table t1;
    exec sql disconnect current;
    return 0;
}
```

The following is an example of using a host variable to specify connection parameters:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
    /* The values of dbname, user, and pwd must be read from environment variables or configuration files.
Environment variables need to be configured as required. If no environment variable is used, a character
string can be directly assigned. */
    char *dbname     = getenv("EXAMPLE_DBNAME_ENV");    /* Database name */
    char *user       = getenv("EXAMPLE_USERNAME_ENV");  /* Username for connection */
    char *pwd        = getenv("EXAMPLE_PASSWD_ENV"); /* Password */
    char *connection = "tcp:gaussdb://localhost:$PORT/testdb";  /* Connection string */
    char ver[256];                  /* Buffer for storing version strings */
EXEC SQL END DECLARE SECTION;

    ECPGdebug(1, stderr);
    EXEC SQL CONNECT TO :dbname;
```



```
    EXEC SQL SELECT pg_catalog.set_config('search_path', '', false); EXEC SQL COMMIT;
    EXEC SQL SELECT version() INTO :ver;
    EXEC SQL DISCONNECT;

    printf("version: %s\n", ver);
    EXEC SQL CONNECT TO :connection USER :user USING :pwd;
    EXEC SQL SELECT pg_catalog.set_config('search_path', '', false); EXEC SQL COMMIT;
    EXEC SQL SELECT version() INTO :ver;
    EXEC SQL DISCONNECT;

    printf("version: %s\n", ver);
    return 0;
}
```

Helpful Links

DISCONECT and **SET CONNECTION**

5.8.5.5.3 DEALLOCATE DESCRIPTOR

Function

Deallocates a SQL descriptor area.

Syntax

```
DEALLOCATE DESCRIPTOR name
```

Parameters

name

Name of an SQL descriptor, case sensitive. This can be an SQL identifier or a host variable.

Examples

```
DEALLOCATE DESCRIPTOR mydesc;
```

Helpful Links

DEALLOCATE DESCRIPTOR, **GET DESCRIPTOR**, **SET DESCRIPTOR**

5.8.5.5.4 DECLARE

Description

Declares a cursor for iterating over the result set of a prepared statement. This command is slightly semantically different from the SQL command **DECLARE**: Whereas the latter executes a query and prepares the result set for retrieval, this embedded SQL command merely declares a name as a "loop variable" for iterating over the result set of a query; the actual execution happens when the cursor is opened with the **OPEN** command.

Syntax

```
DECLARE cursor_name [ BINARY ]  [ NO SCROLL ] CURSOR [ { WITH | WITHOUT } HOLD ] FOR prepared_name
DECLARE cursor_name [ BINARY ]  [ NO SCROLL ] CURSOR [ { WITH | WITHOUT } HOLD ] FOR query
```

Parameters

- **cursor_name**

  Cursor name, which is case sensitive. It can be an SQL identifier or a host variable.

- **prepared_name**

  Name of a prepared query. It can be an SQL identifier or a host variable.

- **query**

  A SELECT command for providing the rows to be returned by the cursor.

NOTE

For details about the cursor options, see **DECLARE**.

Examples

Examples of declaring a cursor used for query:

```
EXEC SQL DECLARE C CURSOR FOR SELECT * FROM My_Table;
EXEC SQL DECLARE C CURSOR FOR SELECT Item1 FROM T;
EXEC SQL DECLARE cur1 CURSOR FOR SELECT version();
```

Example of declaring a cursor for a prepared statement:

```
EXEC SQL PREPARE stmt1 AS SELECT version();
EXEC SQL DECLARE cur1 CURSOR FOR stmt1;
```

Helpful Links

OPEN

5.8.5.5.5 DESCRIBE

Function

Retrieves metadata information for the result columns contained in prepared statements.

Syntax

```
DESCRIBE [ OUTPUT ] prepared_name USING  SQL  DESCRIPTOR descriptor_name
DESCRIBE [ OUTPUT ] prepared_name INTO SQL  DESCRIPTOR descriptor_name
DESCRIBE [ OUTPUT ] prepared_name INTO sqlda_name
```

Parameters

- **prepared_name**

  Name of a prepared statement, which can be an SQL identifier or a host variable.

- **descriptor_name**

  Descriptor name, which is case sensitive. It can be an SQL identifier or a host variable.

- **sqlda_name**

  Name of an SQLDA variable.

Examples

```
EXEC SQL ALLOCATE DESCRIPTOR mydesc;
EXEC SQL PREPARE stmt1 FROM :sql_stmt;
EXEC SQL DESCRIBE stmt1 INTO SQL DESCRIPTOR mydesc;
EXEC SQL GET DESCRIPTOR mydesc VALUE 1 :charvar = NAME;
EXEC SQL DEALLOCATE DESCRIPTOR mydesc;
```

Helpful Links

ALLOCATE DESCRIPTOR, **GET DESCRIPTOR**

5.8.5.5.6 DISCONECT

Description

Closes one or all database connections.

Syntax

```
DISCONNECT connection_name
DISCONNECT [ CURRENT ]
DISCONNECT DEFAULT
DISCONNECT ALL
```

Parameters

- **connection_name**

  Specifies the name of the database connection established by the **CONNECT** command.

- **current**

  Closes the current connection, which can be a recently opened connection or a connection set by the **SET CONNECTION** command. This is also the default if no parameter is passed to the **DISCONNECT** command.

- **default**

  Closes the default connection.

- **all**

  Closes all open connections.

Examples

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
    /* Create the testdb database in advance. */
    EXEC SQL CONNECT TO testdb AS DEFAULT;
    EXEC SQL CONNECT TO testdb AS con1;
    EXEC SQL CONNECT TO testdb AS con2;
    EXEC SQL CONNECT TO testdb AS con3;
    EXEC SQL DISCONNECT CURRENT;  /* Close connection 3. */
    EXEC SQL DISCONNECT DEFAULT;  /* Close the default connection. */
```



```
    EXEC SQL DISCONNECT ALL;     /* Close connections 2 and 1. */
    return 0;
}
```

Helpful Links

CONNECT, **SET CONNECTION**

5.8.5.5.7 EXECUTE IMMEDIATE

Function

Immediately prepares and executes a dynamically specified SQL statement without retrieving result rows.

Syntax

```
EXECUTE IMMEDIATE string
```

Parameters

string

A C string or host variable that contains the SQL statement to be executed.

Examples

The following is an example of executing the INSERT statement using EXECUTE IMMEDIATE and a host variable named **command**:

```
sprintf(command, "INSERT INTO test (name, amount, letter) VALUES ('db: ''r1''', 1, 'f')");
EXEC SQL EXECUTE IMMEDIATE :command;
```

5.8.5.5.8 GET DESCRIPTOR

Description

Retrieves information about a query result set and stores it into host variables. A descriptor area is typically populated using FETCH or SELECT before using this command to transfer the information into host language variables. This command can be in either of the following formats:

- Retrieves the descriptor "header" items, which applies to the result set in its entirety.
- Retrieves information about a particular column, requiring the column number as additional parameter.

Syntax

```
GET DESCRIPTOR descriptor_name VALUE column_number :cvariable = descriptor_item [, ... ]
GET DESCRIPTOR descriptor_name:cvariable = descriptor_header_item [, ... ]
```

Parameters

- **descriptor_name**

  Descriptor name.

- **descriptor_header_item**

  Header item to be retrieved. Currently, only COUNT that is used to obtain the number of columns in the result set is supported.

- **column_number**

  Number of the column about which information is to be retrieved. The count starts at 1.

- **descriptor_item**

  Information item about a column to be retrieved.

- **cvariable**

  A host variable that will receive the data retrieved from the descriptor area.

Examples

Retrieve the number of columns in a result set.

```
EXEC SQL GET DESCRIPTOR d :d_count = COUNT;
```

Retrieve the data length in the first column.

```
EXEC SQL GET DESCRIPTOR d VALUE 1 :d_returned_octet_length = RETURNED_OCTET_LENGTH;
```

Retrieve the data body of the second column as a string.

```
EXEC SQL GET DESCRIPTOR d VALUE 2 :d_data = DATA;
```

Run **SELECT current_database();**. The number of columns, column data length, and column data are displayed.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
    int  d_count = 0;
    char d_data[1024] = {0};
    int  d_returned_octet_length = 0;
EXEC SQL END DECLARE SECTION;
   /* Create testdb in advance. */
    EXEC SQL CONNECT TO testdb;
    EXEC SQL SELECT pg_catalog.set_config('search_path', '', false); EXEC SQL COMMIT;
    EXEC SQL ALLOCATE DESCRIPTOR d;

    /* Declare and open a cursor, and allocate a descriptor to the cursor. */
    EXEC SQL DECLARE cur CURSOR FOR SELECT current_database();
    EXEC SQL OPEN cur;
    EXEC SQL FETCH NEXT FROM cur INTO SQL DESCRIPTOR d;

    /* Obtain the total number of columns. */
    EXEC SQL GET DESCRIPTOR d :d_count = COUNT;
    printf("d_count = %d\n", d_count);

    /* Obtain the length of a returned column. */
    EXEC SQL GET DESCRIPTOR d VALUE 1 :d_returned_octet_length = RETURNED_OCTET_LENGTH;
    printf("d_returned_octet_length = %d\n", d_returned_octet_length);

    /* Fetch the returned column as a string. */
    EXEC SQL GET DESCRIPTOR d VALUE 1 :d_data = DATA;
    printf("d_data = %s\n", d_data);

    /* Closed */
    EXEC SQL CLOSE cur;
    EXEC SQL COMMIT;

    EXEC SQL DEALLOCATE DESCRIPTOR d;
```



```
    EXEC SQL DISCONNECT ALL;
    return 0;
}
```

The following is the command output:

```
d_count              = 1
d_returned_octet_length = 6
d_data               = testdb
```

Helpful Links

ALLOCATE DESCRIPTOR, **DEALLOCATE DESCRIPTOR**, **SET DESCRIPTOR**

5.8.5.5.9 OPEN

Description

Opens a cursor and optionally binds actual values to placeholders in the cursor declaration. The cursor must have been declared using the DECLARE command. Executing the OPEN command triggers the query on the server.

Syntax

```
OPEN cursor_name
OPEN cursor_name USING value [, ... ]
OPEN cursor_name USING SQL DESCRIPTOR descriptor_name
```

Parameters

- **cursor_name**

  Name of the cursor to be opened. It can be an SQL identifier or a host variable.

- **value**

  A value that is to be bound to a placeholder in the cursor declaration. It can be an SQL constant, a host variable, or a host variable with an indicator.

- **descriptor_name**

  Name of the descriptor that contains the value to be bound to the placeholder in the cursor declaration. It can be an SQL identifier or a host variable.

Examples

```
EXEC SQL OPEN a;
EXEC SQL OPEN d USING 1, 'test';
EXEC SQL OPEN c1 USING SQL DESCRIPTOR mydesc;
EXEC SQL OPEN :curname1;
```

Helpful Links

DECLARE

5.8.5.5.10 PREPARE

Description

Prepares the statement to be executed.

Syntax

```
PREPARE name FROM string
```

Parameters

- **name**

  An identifier for the prepared query.

- **string**

  A C string or host variable that contains a prepared statement, which can be SELECT, INSERT, UPDATE, or DELETE.

Examples

```
char *stmt = "SELECT * FROM test1 WHERE a = ? AND b = ?";
EXEC SQL ALLOCATE DESCRIPTOR outdesc;
EXEC SQL PREPARE foo FROM :stmt;
EXEC SQL EXECUTE foo USING SQL DESCRIPTOR indesc INTO SQL DESCRIPTOR outdesc;
```

NOTICE

The PREPARE statement provided by ecpg is not equivalent to the PREPARE syntax provided by the kernel. Example:

GaussDB Kernel kernel syntax:

```
PREPARE name [ ( data_type [, ...] ) ] AS statement
```

Embedded SQL statement:

```
EXEC SQL PREPARE I (int, int) AS INSERT INTO T VALUES ( $1, $2 );
EXEC SQL EXECUTE I(1, 2);
```

When the preceding statement is executed, an error message "too few arguments on" is reported. ecpg provides a dynamic SQL statement to solve the problem in the **PREPARE name [(data_type [, ...])] AS statement** syntax scenario.

Example of using dynamic SQL syntax rules to solve the preceding problem:

```
EXEC SQL PREPARE I AS INSERT INTO T VALUES ( $1, $2 );
EXEC SQL EXECUTE I using 1, 2;
```

5.8.5.5.11 SET AUTOCOMMIT

Function

Sets the autocommit behavior of the current database session. By default, embedded SQL programs do not automatically commit, so you need to explicitly issue COMMIT. This command can change the session to the automatic commit mode so that each individual statement is implicitly committed.

Syntax

```
SET AUTOCOMMIT { = | TO } { ON | OFF }
```

5.8.5.5.12 SET CONNECTION

Function

Sets a database connection.

Syntax

```
SET CONNECTION [ TO | = ] connection_name
```

Parameters

- **connection_name**

  Name of a database connection established by the CONNECT command.

Examples

```
EXEC SQL SET CONNECTION TO con2;
EXEC SQL SET CONNECTION = con1;
```

Helpful Links

CONNECT, **DISCONECT**

5.8.5.5.13 SET DESCRIPTOR

Description

Populates an SQL descriptor area, which is usually used to bind parameters in a prepared query execution. This command can be in either of the following formats:

- Applies to the descriptor "header", which is independent of specific data.
- Assigns a value to specific data identified by a number.

Syntax

```
SET DESCRIPTOR descriptor_name descriptor_header_item = value [, ... ]
SET DESCRIPTOR descriptor_name VALUE number descriptor_item = value [, ...]
```

Parameters

- **descriptor_name**

  SQL descriptor name.

- **descriptor_header_item**

  Identifies the header information item to be set. Currently, only COUNT that can be used to set the number of descriptor items is supported.

- **number**

  Number of descriptor items to be set. The count starts at 1.

- **descriptor_item**

  Identifies which descriptor item is to be set. Currently, only DATA, TYPE, and LENGTH are supported.

- **value**

  Value to be stored in the descriptor item. The value can be an SQL constant or a host variable.

Examples

```
EXEC SQL SET DESCRIPTOR indesc COUNT = 1;
EXEC SQL SET DESCRIPTOR indesc VALUE 1 DATA = 2;
```



```
EXEC SQL SET DESCRIPTOR indesc VALUE 1 DATA = :val1;
EXEC SQL SET DESCRIPTOR indesc VALUE 2 INDICATOR = :val1, DATA = 'some string';
EXEC SQL SET DESCRIPTOR indesc VALUE 2 INDICATOR = :val2null, DATA = :val2;
```

Helpful Links

ALLOCATE DESCRIPTOR, **DEALLOCATE DESCRIPTOR**, and **GET DESCRIPTOR**

5.8.5.5.14 TYPE

Function

Defines a new data type. This command is identified only when ecpg is run with the **-c** option.

Syntax

```
TYPE type_name IS ctype
```

Parameters

type_name

Data type name.

ctype

C type description.

Examples

```
EXEC SQL TYPE customer IS
   struct
   {
      varchar name[50];
      int     phone;
   };

EXEC SQL TYPE cust_ind IS
   struct ind
   {
      short   name_ind;
      short   phone_ind;
   };

EXEC SQL TYPE c IS char reference;
EXEC SQL TYPE ind IS union { int integer; short smallint; };
EXEC SQL TYPE intarray IS int[AMOUNT];
EXEC SQL TYPE str IS varchar[BUFFERSIZ];
EXEC SQL TYPE string IS char[11];
```

Example of using EXEC SQL TYPE (note that the -c parameter needs to be added in the ecpg preprocessing phase when the following example is used):

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

EXEC SQL WHENEVER SQLERROR SQLPRINT;
EXEC SQL TYPE tt IS
   struct
   {
      varchar v[256];
      int     i;
```



```
    };
EXEC SQL TYPE tt_ind IS
    struct ind {
        short   v_ind;
        short   i_ind;
    };

int main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
    tt t;
    tt_ind t_ind;
EXEC SQL END DECLARE SECTION;
     EXEC SQL CONNECT TO testdb AS con1;
     EXEC SQL SELECT current_database(), 256 INTO :t:t_ind LIMIT 1;

     printf("t.v = %s\n", t.v.arr);
     printf("t.i = %d\n", t.i);

     printf("t_ind.v_ind = %d\n", t_ind.v_ind);
     printf("t_ind.i_ind = %d\n", t_ind.i_ind);

     EXEC SQL DISCONNECT con1;

     return 0;
}
```

The output of this example is as follows.

```
t.v = testdb
t.i = 256
t_ind.v_ind = 0
t_ind.i_ind = 0
```

5.8.5.5.15 VAR

Function

Assigns a new C data type to a host variable. The host variable must have been declared in a DECLARE segment.

NOTE

- Exercise caution when using VAR. Using VAR to change the data type my cause the memory address to be invalid. As a result, the data variable is invalid and the value cannot be assigned.
- If the data type has been defined in the host variable DECLARE segment, you do not need to use the VAR statement.

Syntax

```
VAR varname IS ctype
```

Parameters

- **varname**

  Name of a C variable.
- **ctype**

  C type description.

Examples

```
EXEC SQL BEGIN DECLARE SECTION;
    short a;
```

```
EXEC SQL END DECLARE SECTION;
EXEC SQL VAR a IS int;
```

5.8.5.5.16 WHENEVER

Description

Defines a behavior that is invoked when an SQL execution exception occurs (row not found, SQL alarm, or error).

Syntax

```
WHENEVER { NOT FOUND | SQLERROR | SQLWARNING } action
```

Parameters

For details about the parameter description, see **Setting Callbacks**.

Examples

```
EXEC SQL WHENEVER NOT FOUND CONTINUE;
EXEC SQL WHENEVER NOT FOUND DO BREAK;
EXEC SQL WHENEVER SQLWARNING SQLPRINT;
EXEC SQL WHENEVER SQLWARNING DO warn();
EXEC SQL WHENEVER SQLERROR sqlprint;
EXEC SQL WHENEVER SQLERROR SQLCALL print2();
EXEC SQL WHENEVER SQLERROR DO handle_error("select");
EXEC SQL WHENEVER SQLERROR DO sqlnotice(NULL, NONO);
EXEC SQL WHENEVER SQLERROR DO sqlprint();
EXEC SQL WHENEVER SQLERROR GOTO error_label;
EXEC SQL WHENEVER SQLERROR STOP;
```

Use WHENEVER NOT FOUND BREAK to process the loop of the result set. The following is a complete example:

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int main(void)
{
    EXEC SQL CONNECT TO testdb AS con1;
    EXEC SQL ALLOCATE DESCRIPTOR d;
    EXEC SQL DECLARE cur CURSOR FOR SELECT current_database(), 'hoge', 256;
    EXEC SQL OPEN cur;
    EXEC SQL BEGIN DECLARE SECTION;
    char* d1;
    char* d2;
    EXEC SQL END DECLARE SECTION;
    /* When the end of the result set is reached, exit the loop. */
    EXEC SQL WHENEVER NOT FOUND DO BREAK;

    while (1)
    {
        EXEC SQL FETCH NEXT FROM cur INTO SQL DESCRIPTOR d;
        exec sql get descriptor d value 1 :d1=DATA;
        exec sql get descriptor d value 2 :d2=DATA;
        printf("d1 is %s,%s\n", d1, d2) ;
    }
     EXEC SQL CLOSE cur;
     EXEC SQL COMMIT;
     EXEC SQL DEALLOCATE DESCRIPTOR d;
     EXEC SQL DISCONNECT ALL;
     return 0;
}
```


5.8.6 Querying the Result Set

- The SELECT statement that returns the result of a single row can be directly executed using EXEC SQL. For details, see **Executing SQL Statements**.

  Example:

```
/* Create a table and insert data. */
EXEC SQL CREATE TABLE test_table (number1 integer, number2 integer);
EXEC SQL INSERT INTO test_table (number1, number2) VALUES (2, 1);

/* The query result is in a single row. :num is the host variable. */
EXEC SQL SELECT number1 INTO :num FROM test_table WHERE number2 = 1;
```

- To process a multi-row result set, you must use a cursor. For details, see **Using Cursors**. (In special cases, an application can fetch multiple rows of results at a time and write them to the host variable of the array type. For details, see **Host Variables with Non-Primitive Types**.)

  Example:

```
    /* Create a table and insert data. */
    EXEC SQL CREATE TABLE test_table (number1 integer, number2 integer);
    EXEC SQL INSERT INTO test_table (number1, number2) VALUES (2, 1);
    EXEC SQL INSERT INTO test_table (number1, number2) VALUES (3, 1);
    EXEC SQL INSERT INTO test_table (number1, number2) VALUES (4,1);
    EXEC SQL INSERT INTO test_table (number1, number2) VALUES (5, 1);

/* Define the host variable. */
EXEC SQL BEGIN DECLARE SECTION;
    int v1;
    int v2;
EXEC SQL END DECLARE SECTION;

    /* Declare a cursor. */
    EXEC SQL DECLARE test_bar CURSOR FOR SELECT number1, number2 FROM test_table ORDER BY
number1;
    /* Open the cursor. */
    EXEC SQL OPEN test_bar;
    /* When the cursor reaches the end of the result set, exit the loop. */
    EXEC SQL WHENEVER NOT FOUND DO BREAK;
    /* Obtain the query result set. */
    while(1)
    {
        EXEC SQL FETCH NEXT FROM test_bar INTO :v1, :v2;
        printf("number1 = %d, number2 = %d\n",v1,v2);
    }
    /* Close the cursor. */
    EXEC SQL CLOSE test_bar;
```

5.8.7 Closing a Database Connection

Close the database connection after the database is used.

Close a connection.

```
EXEC SQL DISCONNECT [connection];
```

Declare the connection by using the following methods:

- connection-name
- default
- current
- all

5.8.8 Host Variables

This section describes how to use host variables to pass data between C programs and embedded SQL programs. In C programs with embedded SQL statements, we use the C language as the host language and regard EXEC SQL [Command] statements as embedded SQL statements of the host language. Therefore, variables used for embedded SQL statements in C programs are called host variables.

5.8.8.1 Overview

Passing data between a C program and SQL statements is particularly simple in embedded SQL. Instead of having the program paste data into the statements, you can simply write the name of a C variable into the SQL statement, prefixed by a colon. The following is an example.

```
EXEC SQL INSERT INTO sometable VALUES (:v1, 'foo', :v2);
```

This statement references two C variables named v1 and v2 and uses a regular SQL string, indicating that you are not restricted to use one kind of data or the other.

5.8.8.2 DECLARE Section

To implement data interaction between a C program with embedded SQL statements and a database, for example, to pass parameters in a query of the C program to the database, or to pass data from the database back to the program, the C variables that are intended to contain this data need to be declared in specially marked sections, so that the embedded SQL preprocessor is made aware of them.

This section starts with:

```
EXEC SQL BEGIN DECLARE SECTION;
```

And ends with:

```
EXEC SQL END DECLARE SECTION;
```

Between them, there must be regular C variable declarations. The following is an example.

```
int   x = 4;
char  foo[16], bar[16];
```

NOTICE

- The type of the host variables declared between the start and end of the marked section must be one of the supported data types. For details, see **Table 5-90**.
- You can also declare variables with the following syntax which implicitly creates a DECLARE section: EXEC SQL int i = 4.
- Variables that are not intended to be used in SQL commands can be declared normally outside these special sections.
- The definition of a structure or union also must be listed inside a DECLARE section. Otherwise, the preprocessor cannot process these types.

5.8.8.3 Retrieving Query Results

To retrieve the results of a query, embedded SQL provides special variants of the usual commands SELECT and FETCH. These commands have a special INTO clause that specifies which host variables the retrieved values are to be stored in. SELECT is used for a query that returns only a single row, and FETCH is used for a query that returns multiple rows, using a cursor.

- Here is an example using the command SELECT:

```
/*
 * Assume a table:
 * CREATE TABLE test1 (a int, b varchar(50));
 */
EXEC SQL BEGIN DECLARE SECTION;
    int v1;
    VARCHAR v2;
EXEC SQL END DECLARE SECTION;

    ...

    EXEC SQL SELECT a, b INTO :v1, :v2 FROM test;
```

  The INTO clause appears between the SELECT list and the FROM clause. The number of elements in the SELECT list and the list after INTO (also called the target list) must be equal.

- Here is an example using the command **FETCH**:

```
EXEC SQL BEGIN DECLARE SECTION;
    int v1;
    VARCHAR v2;
EXEC SQL END DECLARE SECTION;
    ...

    EXEC SQL DECLARE foo CURSOR FOR SELECT a, b FROM test;
    ...
    do
    {
        ...
        EXEC SQL FETCH NEXT FROM foo INTO :v1, :v2;
        ...
    } while (...);
```

  The INTO clause appears after all SQL clauses.

5.8.8.4 Type Mapping

When ecpg applications exchange values between the GaussDB Kernel server and the C program, such as when retrieving query results from the server or executing SQL statements with input parameters, the values need to be converted between GaussDB Kernel data types and host language variable types (C language data types, concretely). There are two data types available: Simple GaussDB Kernel data types, such as integer and text, can be directly read and written by applications. Other GaussDB Kernel data types, such as timestamp and numeric, can be accessed only by using special library functions. For details, see **ECPG API Reference**.

Table 5-90 Mapping between GaussDB Kernel data types and C variable types

GaussDB Kernel Data Type	Host Variable Type
smallint	short

GaussDB Kernel Data Type	Host Variable Type
integer	int
bigint	long long int
boolean	boolean
character(*n*), varchar(n), text	char[n+1], VARCHAR[n+1]
double precision	double
real	float
smallserial	short
serial	int
bigserial	long long int
oid	unsigned int
name	char[NAMEDATALEN]
date	date [a]
timestamp	timestamp [a]
interval	interval [a]
decimal	decimal [a]
numeric	numeric [a]

NOTE

[a] This type can be accessed through **Accessing Special Data Types**.

NOTICE

- Currently, only basic data types of the C language can be used or combined. The string data type in the C++ language cannot be used as the host variable type.
- Currently, ecpg maps only common data types of GaussDB Kernel SQL. For details about the supported data types, see **Table 5-90**.

5.8.8.5 Handling Character Strings

To handle SQL character string data types, such as varchar and text, there are two possible methods to declare the host variables.

1. Method 1: Use char[] (a char string), which is the most common method for processing character data in C programs.


```
EXEC SQL BEGIN DECLARE SECTION;
   char str[50];
EXEC SQL END DECLARE SECTION;
```

Note that you have to take care of the length yourself. If you use this host variable as the target variable of a query which returns a string with more than 49 characters, a buffer overflow occurs.

2. Method 2: Use the VARCHAR type, which is a special type provided by ECPG. The definition on an array of type VARCHAR is converted into a named struct for every variable. The following is a declaration example.

```
VARCHAR var[180];
```

It will be converted into:

```
struct varchar_var
{
   int len;
   char arr[180];
} var;
```

To store a string in a VARCHAR host variable, the host variable has to be declared as a string including the zero-byte terminator. **arr** stores the string including a terminating zero byte. **len** stores the length of the string stored in **arr** without the terminating zero byte. The terminator is not included when the length is calculated. When a host variable is used as input for a query, if the values of **strlen(arr)** and **len** are different, the shorter one is used.

CAUTION

- VARCHAR can be written in upper or lower case, but not in mixed case.
- char and VARCHAR host variables can also hold values of other SQL types, which will be stored in their string forms.

5.8.8.6 Host Variables with Non-Primitive Types

Non-primitive host variables can be arrays, typedefs, structures, and pointers.

- Arrays

  There are two use cases for arrays as host variables. The first case is to store some text strings in char[] or VARCHAR[]. The second case is to retrieve multiple rows from a query result without using a cursor. Without an array, to process a query result consisting of multiple rows, it is required to use a cursor and the FETCH command. But with array host variables, multiple rows can be received at once. The length of the array has to be defined to be able to accommodate all rows, otherwise a buffer overflow will occur.

  The following is an example of scanning the pg_database system catalog and displays the OIDs and names of all available databases.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
   int dbid[8];
   char dbname[8][16];
   int i;
EXEC SQL END DECLARE SECTION;
```



```
    memset(dbname, 0, sizeof(char)* 16 * 8);
    memset(dbid, 0, sizeof(int) * 8);
  /* Connect to the testdb database. The testdb database must be created in advance. */
    EXEC SQL CONNECT TO testdb;
    /* Retrieve multiple rows to arrays at a time. */
    EXEC SQL SELECT oid,datname INTO :dbid, :dbname FROM pg_database;
    for (i = 0; i < 8; i++)
        printf("oid=%d, dbname=%s\n", dbid[i], dbname[i]);
    EXEC SQL COMMIT;
    EXEC SQL DISCONNECT ALL;
    return 0;
}
```

The following is the example output (the exact value depends on the local environment).

```
oid=1, dbname=template1
oid=11510, dbname=template0
oid=11511, dbname=postgres
oid=313780, dbname=testdb
oid=0, dbname=
oid=0, dbname=
oid=0, dbname=
```

- Structures

  A structure whose member names match the column names of a query result, can be used to retrieve multiple columns at once. The structure enables handling multiple column values in a single host variable.

  The following example retrieves OIDs, names, and sizes of the available databases from the pg_database system catalog and using the pg_database_size() function. In this example, a structure variable **dbinfo_t** with members whose names match each column in the SELECT result is used to retrieve one result row without putting multiple host variables in the FETCH statement.

```
EXEC SQL BEGIN DECLARE SECTION;
    typedef struct
    {
        int oid;
        char datname[65];
        long long int size;
    } dbinfo_t;

    dbinfo_t dbval;
EXEC SQL END DECLARE SECTION;
    memset(&dbval, 0, sizeof(dbinfo_t));

    EXEC SQL DECLARE cur1 CURSOR FOR SELECT oid, datname, pg_database_size(oid) AS size FROM
pg_database;
    EXEC SQL OPEN cur1;

    /* Exit the while loop when the end of the result set is reached. */
    EXEC SQL WHENEVER NOT FOUND DO BREAK;

    while (1)
    {
        /* Fetch multiple columns into one structure. */
        EXEC SQL FETCH FROM cur1 INTO :dbval;
        /* Print the members of the structure. */
        printf("oid=%d, datname=%s, size=%lld\n", dbval.oid, dbval.datname, dbval.size);
    }
    EXEC SQL CLOSE cur1;
```

The following is the example output (the exact value depends on the local environment).

```
oid=1, datname=template1, size=4324580
oid=11510, datname=template0, size=4243460
```



```
oid=11511, datname=postgres, size=4324580
oid=313780, datname=testdb, size=8183012
```

Structure host variables "absorb" as many columns in the query result as the structure columns. Additional columns can be allocated to other host variables. The above program could also be restructured like this, with the **size** variable outside the structure:

```
EXEC SQL BEGIN DECLARE SECTION;
   typedef struct
   {
      int oid;
      char datname[65];
   } dbinfo_t;

   dbinfo_t dbval;
   long long int size;
EXEC SQL END DECLARE SECTION;

   memset(&dbval, 0, sizeof(dbinfo_t));

   EXEC SQL DECLARE cur1 CURSOR FOR SELECT oid, datname, pg_database_size(oid) AS size FROM
pg_database;
   EXEC SQL OPEN cur1;

   /* Exit the while loop when the end of the result set is reached. */
   EXEC SQL WHENEVER NOT FOUND DO BREAK;
   while (1)
   {
      /* Fetch multiple columns into one structure. */
      EXEC SQL FETCH FROM cur1 INTO :dbval, :size;
      /* Print the members of the structure. */
      printf("oid=%d, datname=%s, size=%lld\n", dbval.oid, dbval.datname, size);
   }

   EXEC SQL CLOSE cur1;
```

- typedef

  Use the typedef keyword to map new types to existing types.

  ```
  EXEC SQL BEGIN DECLARE SECTION;
  typedef char mychartype[40];
  typedef long serial_t;
  EXEC SQL END DECLARE SECTION;
  ```

  You can also run the following command:

  ```
  EXEC SQL TYPE serial_t IS long;
  ```

  This declaration does not need to be part of a DECLARE section.

- Pointers

  You can declare pointers to the most common types.

  ```
  EXEC SQL BEGIN DECLARE SECTION;
  int   *intp;
  char **charp;
  EXEC SQL END DECLARE SECTION;
  ```

5.8.8.7 Accessing Special Data Types

ECPG supports the numeric, decimal, date, timestamp, and interval data types. These data types cannot be mapped to primitive host variable types because they have a complex internal structure. Applications deal with these types by declaring host variables in special types and accessing them using functions in the pgtypes library. For details about the functions in the pgtypes library, see **ECPG API Reference**.

- timestamp and date

  First, the program must include the header file for the timestamp type.


```
#include <pgtypes_timestamp.h>
```

Then, declare a host variable of the timestamp type in the DECLARE section.

```
EXEC SQL BEGIN DECLARE SECTION;
    timestamp ts;
EXEC SQL END DECLARE SECTION;
```

After the value is read to the host variable, the pgtypes library function is used for processing. In the following example, the PGTYPEStimestamp_to_asc() function is used to convert the timestamp value to the text format:

```
EXEC SQL SELECT now()::timestamp INTO :ts;
printf("ts = %s\n", PGTYPEStimestamp_to_asc(ts));
```

A command output example is as follows:

```
ts = 2022-06-27 18:03:56.949343
```

In addition, the date type can be processed in the same way. The program must contain the **pgtypes_date.h** header file, declare a host variable as the date type, and use the PGTYPEdata_to_asc() function to convert the variable to the text format.

- interval

  The handling of the interval type is also similar to the timestamp and date types. However, to allocate memory for an interval type value explicitly, the memory space for the variable must be allocated from the heap memory.

  For example:

  ```
  #include <stdio.h>
  #include <stdlib.h>
  #include <pgtypes_interval.h>
  int main(void)
  {
  EXEC SQL BEGIN DECLARE SECTION;
      interval *in;
  EXEC SQL END DECLARE SECTION;
      /* Connect to the testdb database. The testdb database must be created in advance.*/
      EXEC SQL CONNECT TO testdb;
      in = PGTYPESinterval_new();
      EXEC SQL SELECT '1 min'::interval INTO :in;
      printf("interval = %s\n", PGTYPESinterval_to_asc(in));
      PGTYPESinterval_free(in);
      EXEC SQL COMMIT;
      EXEC SQL DISCONNECT ALL;
      return 0;
  }
  ```

- numeric and decimal

  The handling of the numeric and decimal types is similar to the interval type: It requires defining a pointer, allocating some memory space from the heap, and accessing the variable using the pgtypes library functions.

  Example:

  ```
  #include <stdio.h>
  #include <stdlib.h>
  #include <pgtypes_numeric.h>
  EXEC SQL WHENEVER SQLERROR STOP;
  int main(void)
  {
  EXEC SQL BEGIN DECLARE SECTION;
      numeric *num;
      numeric *num2;
      decimal *dec;
  EXEC SQL END DECLARE SECTION;
      /* Connect to the testdb database. The testdb database must be created in advance.*/
      EXEC SQL CONNECT TO testdb;

      num = PGTYPESnumeric_new();
  ```



```
    dec = PGTYPESdecimal_new();

    EXEC SQL SELECT 12.345::numeric(4,2), 23.456::decimal(4,2) INTO :num, :dec;
    printf("numeric = %s\n", PGTYPESnumeric_to_asc(num, 0));
    printf("numeric = %s\n", PGTYPESnumeric_to_asc(num, 1));
    printf("numeric = %s\n", PGTYPESnumeric_to_asc(num, 2));
    /* Convert decimal to numeric to show a decimal value. */
    num2 = PGTYPESnumeric_new();
    PGTYPESnumeric_from_decimal(dec, num2);
    printf("decimal = %s\n", PGTYPESnumeric_to_asc(num2, 0));
    printf("decimal = %s\n", PGTYPESnumeric_to_asc(num2, 1));
    printf("decimal = %s\n", PGTYPESnumeric_to_asc(num2, 2));
    PGTYPESnumeric_free(num2);
    PGTYPESdecimal_free(dec);
    PGTYPESnumeric_free(num);

    EXEC SQL COMMIT;
    EXEC SQL DISCONNECT ALL;
    return 0;
}
```

5.8.8.8 Handling Non-Primitive SQL Data Types

This section describes how to handle non-scalar and user-defined SQL-level data types in ECPG applications. Note that this is distinct from the handling of host variables of non-primitive types described in **Host Variables with Non-Primitive Types**.

- Arrays

  Multi-dimensional SQL-level arrays are not directly supported in ECPG. One-dimensional SQL-level arrays can be mapped into C array host variables and vice-versa. However, when creating a statement, ECPG does not know the types of the columns, so that it cannot check if a C array is input into a corresponding SQL-level array. When processing the output of an SQL statement, ECPG has to check if both are arrays.

  If a query accesses elements of an array separately, a host variable with a type that can be mapped to the element type should be used. For example, if a column type is array of integer, a host variable of type int can be used. Also if the element type is varchar or text, a host variable of type char[] or VARCHAR[] can be used.

  For example:

```
CREATE TABLE t3 (
    ii integer[]
);
testdb=> SELECT * FROM t3;
     ii
-------------
 {1,2,3,4,5}
(1 row)
```

  The following example retrieves the fourth element of an array and stores it in a host variable of the int type:

```
EXEC SQL BEGIN DECLARE SECTION;
    int ii;
EXEC SQL END DECLARE SECTION;

    EXEC SQL DECLARE cur1 CURSOR FOR SELECT ii[4] FROM t3;
    EXEC SQL OPEN cur1;

    EXEC SQL WHENEVER NOT FOUND DO BREAK;

    while (1)
    {
```



```
    EXEC SQL FETCH FROM cur1 INTO :ii ;
    printf("ii=%d\n", ii);
  }
  EXEC SQL CLOSE cur1;
```

Example output:

```
ii=4
```

To map multiple array elements to the multiple elements in an array type host variable, each element of the array column and each element of the host variable array must be managed separately. For example:

```
EXEC SQL BEGIN DECLARE SECTION;
  int ii_a[8];
EXEC SQL END DECLARE SECTION;

  EXEC SQL DECLARE cur1 CURSOR FOR SELECT ii[1], ii[2], ii[3], ii[4] FROM t3;
  EXEC SQL OPEN cur1;

  EXEC SQL WHENEVER NOT FOUND DO BREAK;
  while (1)
  {
    EXEC SQL FETCH FROM cur1 INTO :ii_a[0], :ii_a[1], :ii_a[2], :ii_a[3];
    ...
  }
```

Note:

```
EXEC SQL BEGIN DECLARE SECTION;
  int ii_a[8];
EXEC SQL END DECLARE SECTION;

  EXEC SQL DECLARE cur1 CURSOR FOR SELECT ii FROM t3;
  EXEC SQL OPEN cur1;

  EXEC SQL WHENEVER NOT FOUND DO BREAK;

  while (1)
  {
    /* Error */
    EXEC SQL FETCH FROM cur1 INTO :ii_a;
    ...
  }
```

It does not work out because you cannot map an array type column to an array host variable directly.

- Composite types

  Composite types are not directly supported in ECPG. For example, you cannot declare member variables as date type in a structure. However, you can access each attribute separately or use the external string representation.

  In the following example, each attribute can be accessed separately:

  ```
  CREATE TYPE comp_t AS (intval integer, textval varchar(32));
  CREATE TABLE t4 (compval comp_t);
  INSERT INTO t4 VALUES ( (256, 'GaussDB') );
  ```

  The following program retrieves data from the example table by selecting each attribute of the comp_t type separately:

  ```
  EXEC SQL BEGIN DECLARE SECTION;
    int intval;
    varchar textval[33];
  EXEC SQL END DECLARE SECTION;

    /* Put each element of the composite type column in the SELECT list. */
    EXEC SQL DECLARE cur1 CURSOR FOR SELECT (compval).intval, (compval).textval FROM t4;
    EXEC SQL OPEN cur1;

    EXEC SQL WHENEVER NOT FOUND DO BREAK;
    while (1)
    {
  ```



```
    /* Fetch each element of the composite type column into host variables. */
    EXEC SQL FETCH FROM cur1 INTO :intval, :textval;
    printf("intval=%d, textval=%s\n", intval, textval.arr);
  }
  EXEC SQL CLOSE cur1;
```

The host variables storing values in the **FETCH** command can be gathered into one structure. For more details about the host variables in the structure form, see **Handling Character Strings**. In the following example, the two host variables, *intval* and *textval*, become members of the comp_t structure, and the structure is specified in the **FETCH** command:

```
EXEC SQL BEGIN DECLARE SECTION;
  typedef struct
  {
    int intval;
    varchar textval[33];
  } comp_t;
  comp_t compval;
EXEC SQL END DECLARE SECTION;

  /* Put each element of the composite type column in the SELECT list. */
  EXEC SQL DECLARE cur1 CURSOR FOR SELECT (compval).intval, (compval).textval FROM t4;
  EXEC SQL OPEN cur1;

  EXEC SQL WHENEVER NOT FOUND DO BREAK;
  while (1)
  {
    /* Put all values in the SELECT list into one structure. */
    EXEC SQL FETCH FROM cur1 INTO :compval;
    printf("intval=%d, textval=%s\n", compval.intval, compval.textval.arr);
  }
  EXEC SQL CLOSE cur1;
```

Although a structure is used in the FETCH command, the attribute names in the SELECT clause are specified one by one. This can be enhanced by using a * to ask for all attributes of the composite type value. For example:

```
...
EXEC SQL DECLARE cur1 CURSOR FOR SELECT (compval).* FROM t4;
EXEC SQL OPEN cur1;

EXEC SQL WHENEVER NOT FOUND DO BREAK;

while (1)
{
  /* Put all values in the SELECT list into one structure. */
  EXEC SQL FETCH FROM cur1 INTO :compval;
  printf("intval=%d, textval=%s\n", compval.intval, compval.textval.arr);
}
...
```

In this way, composite types can be mapped into structures, even though ECPG does not understand the composite type itself.

- User-defined base types

  When ECPG uses host variables to store query results, only the data types provided by ECPG are supported. User-defined data types are not supported. Data types created using CREATE TYPE cannot be mapped using host variables.

  However, you can use external string representations and host variables of the char[] or VARCHAR[] type to handle user-defined types.

  The external string representation of that type is (%lf,%lf), which is defined in function complex_in(). The following example inserts complex type values **(1,1)** and **(3,3)** into columns **a** and **b** and then queries them from the table:

```
EXEC SQL BEGIN DECLARE SECTION;
  varchar a[64];
```



```
    varchar b[64];
EXEC SQL END DECLARE SECTION;
    EXEC SQL INSERT INTO test_complex VALUES ('(1,1)', '(3,3)');
    EXEC SQL DECLARE cur1 CURSOR FOR SELECT a, b FROM test_complex;
    EXEC SQL OPEN cur1;

    EXEC SQL WHENEVER NOT FOUND DO BREAK;

    while (1)
    {
        EXEC SQL FETCH FROM cur1 INTO :a, :b;
        printf("a=%s, b=%s\n", a.arr, b.arr);
    }
    EXEC SQL CLOSE cur1;
```

Example output:

```
a=(1,1), b=(3,3)
```

5.8.9 Executing Dynamic SQL Statements

In most cases, the SQL statements executed by an application are known when the application is written. However, in some cases, SQL statements are constructed at runtime or provided by external sources. In these cases, the SQL statements cannot be directly embedded into the C source code, but dynamic SQL statements allow you to call the provided SQL statements in a string variable.

5.8.9.1 Executing Statements Without a Result Set

An example of running the **EXECUTE IMMEDIATE** command is as follows:

```
EXEC SQL BEGIN DECLARE SECTION;
    const char *stmt = "CREATE TABLE test1 (...);";
EXEC SQL END DECLARE SECTION;
    EXEC SQL EXECUTE IMMEDIATE :stmt;
```

EXECUTE IMMEDIATE can be used for SQL statements that do not return a result set, such as DDL, INSERT, UPDATE, and DELETE statements. However, statements for retrieving data, such as SELECT statements, cannot be executed in this way.

5.8.9.2 Executing a Statement with Input Parameters

Prepare a normal statement and execute a specific version of it by replacing its parameters (with question marks). Use the EXECUTE statement to execute the prepared statement by specifying parameters in the USING clause. The following is an example.

```
EXEC SQL BEGIN DECLARE SECTION;
    const char *stmt = "INSERT INTO test1 VALUES(?, ?);";
EXEC SQL END DECLARE SECTION;
    /* PREPARE Prepare a statement for execution. */
    EXEC SQL PREPARE mystmt FROM :stmt;
    ...
    /* Single quotation marks are valid characters. If a character string is used, use double quotation marks.*/
    EXEC SQL EXECUTE mystmt USING 42, 'foobar';

    /* If a prepared statement is no longer used, release it in time. */
    EXEC SQL DEALLOCATE PREPARE name;
```

5.8.9.3 Executing a Statement with a Result Set

EXECUTE can be used to execute SQL statements with a result set. To save the result, add an INTO clause. The following is an example.

```
EXEC SQL BEGIN DECLARE SECTION;
    const char *stmt = "SELECT a, b, c FROM test1 WHERE a > ?";
```



```
    int v1, v2;
    VARCHAR v3[50];
EXEC SQL END DECLARE SECTION;

    EXEC SQL PREPARE mystmt FROM :stmt;
    ...
    EXEC SQL EXECUTE mystmt INTO :v1, :v2, :v3 USING 37;
```

NOTE

The EXECUTE statement supports the INTO and USING clauses.

If a query may return multiple result rows, use cursors. For details about cursors, see **Using Cursors**. Example:

```
EXEC SQL BEGIN DECLARE SECTION;
    char dbaname[128];
    char datname[128];
    char *stmt = "SELECT u.usename as dbaname, d.datname "
                 "  FROM pg_database d, pg_user u "
                 "  WHERE d.datdba = u.usesysid";
EXEC SQL END DECLARE SECTION;

    EXEC SQL CONNECT TO testdb AS con1 USER testuser;

    EXEC SQL PREPARE stmt1 FROM :stmt;

    EXEC SQL DECLARE cursor1 CURSOR FOR stmt1;
    EXEC SQL OPEN cursor1;

    EXEC SQL WHENEVER NOT FOUND DO BREAK;

    while (1)
    {
        EXEC SQL FETCH cursor1 INTO :dbaname,:datname;
        printf("dbaname=%s, datname=%s\n", dbaname, datname);
    }
    EXEC SQL CLOSE cursor1;

    EXEC SQL COMMIT;
    EXEC SQL DISCONNECT ALL;
```

5.8.10 Error Handling

There are two non-exclusive methods to handle errors and warnings in an embedded SQL program:

- Callbacks can be configured to handle errors or warnings using the WHENEVER command.
- Detailed information about the errors or warnings can be obtained from the *sqlca* variable.

5.8.10.1 Setting Callbacks

One simple method to catch errors and warnings is to set a specific action to be executed whenever a particular condition occurs. To set the callback, run the following command:

```
EXEC SQL WHENEVER condition action;
```

condition can be one of the following:

- **SQLERROR**: The specified action is called whenever an error occurs during the execution of an SQL statement.
- **SQLWARNING**: The specified action is called whenever a warning occurs during the execution of an SQL statement.

- **NOT FOUND**: The specified action is called whenever an SQL statement retrieves or affects zero rows.

action can be one of the following:

- **CONTINUE**: ignores the callback error condition and continues the execution. It is usually used to stop a break condition. This is the default value.
- **GOTO** *label*/**GO TO** *label*: jumps to a specified label (using the C goto statement).
- **SQLPRINT**: prints a message to the standard error.
- **STOP**: calls exit(1) to terminate the program.
- **DO BREAK**: executes the C statement BREAK. This statement is used only in loops or switch statements.

For example:

```
/* It prints a simple message when a warning occurs and aborts the program when an error happens. */
EXEC SQL WHENEVER SQLWARNING SQLPRINT;
EXEC SQL WHENEVER SQLERROR STOP;
```

NOTICE

- The statement EXEC SQL WHENEVER is a directive of the SQL preprocessor, not a C statement. The error or warning actions that it sets apply to all embedded SQL statements that appear below the point where the handler is set, unless a different action was set for the same condition between the first EXEC SQL WHENEVER and the SQL statement causing the condition, regardless of the flow of control in the C program. Therefore, neither of the following C programs can achieve the expected effect:

```
/*
 * ERROR
 */
void func()
{
    ...
    if (verbose) {
        EXEC SQL WHENEVER SQLWARNING SQLPRINT;
    }
    ...
    EXEC SQL SELECT ...;
    ...
}
/*
 * ERROR
 */
void func()
{
    ...
    set_error_handler();
    ...
    EXEC SQL SELECT ...;
    ...
}
static void set_error_handler(void)
{
    EXEC SQL WHENEVER SQLERROR STOP;
}
```

- DO BREAK can be used only in the while, for, and switch scenarios. After DO BREAK is used, use the CONTINUE statement to ignore it.

5.8.10.2 sqlca

The embedded SQL interface provides a global variable *sqlca* (short for SQL communication area). *sqlca* covers warnings and errors. If multiple warnings or errors occur during the execution of a statement, then *sqlca* will only contain information about the last one. In a multithreaded program, every thread automatically gets its own copy of *sqlca*.

The data structure is as follows:

```
struct
{
    char sqlcaid[8];
    long sqlabc;
    long sqlcode;
    struct
    {
        int sqlerrml;
        char sqlerrmc[SQLERRMC_LEN];
    } sqlerrm;
    char sqlerrp[8];
    long sqlerrd[6];
    char sqlwarn[8];
    char sqlstate[5];
} sqlca;
```

If no error occurred in the last SQL statement, **sqlca.sqlcode** will be **0** and **sqlca.sqlstate** will be **00000**. If a warning or error occurred, then **sqlca.sqlcode** will be negative and **sqlca.sqlstate** will be different from **00000**. For details about the values of **SQLSTATE** and **SQLCODE**, see **SQLSTATE and SQLCODE**.

If the last SQL statement was successful, then **sqlca.sqlerrd[1]** contains the OID of the processed row, if applicable, and **sqlca.sqlerrd[2]** contains the number of processed or returned rows, if applicable to the command.

In case of an error or warning, **sqlca.sqlerrm.sqlerrmc** will contain a string that describes the error. **sqlca.sqlerrm.sqlerrml** contains the length of the error message that is stored in **sqlca.sqlerrm.sqlerrmc**, that is, the result of **strlen()**. Note that some messages are too long to fit in the fixed-size sqlerrmc array; they will be truncated.

When a warning is generated, **sqlca.sqlwarn[2]** is set to **W**.

The fields **sqlcaid**, **sqlcabc**, **sqlerrp**, and the remaining elements of **sqlerrd** and **sqlwarn** currently contain no useful information.

The following is an example.

```
/* Integrate WHENEVER and sqlca to implement error handling. */
EXEC SQL WHENEVER SQLERROR SQLCALL print_sqlca();

void print_sqlca()
{
    fprintf(stderr, "==== sqlca ====\n");
    fprintf(stderr, "sqlcode: %ld\n", sqlca.sqlcode);
    fprintf(stderr, "sqlerrm.sqlerrml: %d\n", sqlca.sqlerrm.sqlerrml);
    fprintf(stderr, "sqlerrm.sqlerrmc: %s\n", sqlca.sqlerrm.sqlerrmc);
    fprintf(stderr, "sqlerrd: %ld %ld %ld %ld %ld %ld\n", sqlca.sqlerrd[0],sqlca.sqlerrd[1],sqlca.sqlerrd[2],
                                            sqlca.sqlerrd[3],sqlca.sqlerrd[4],sqlca.sqlerrd[5]);
    fprintf(stderr, "sqlwarn: %d %d %d %d %d %d %d %d\n", sqlca.sqlwarn[0], sqlca.sqlwarn[1],
sqlca.sqlwarn[2],
                                            sqlca.sqlwarn[3], sqlca.sqlwarn[4], sqlca.sqlwarn[5],
                                            sqlca.sqlwarn[6], sqlca.sqlwarn[7]);
    fprintf(stderr, "sqlstate: %5s\n", sqlca.sqlstate);
    fprintf(stderr, "===============\n");
}
```


The output is similar to the following (here is a misspelled table name):

```
==== sqlca ====
sqlcode: -400
sqlerrm.sqlerrml: 49
sqlerrm.sqlerrmc: relation "pg_databasep" does not exist on line 38
sqlerrd: 0 0 0 0 0 0
sqlwarn: 0 0 0 0 0 0 0 0
sqlstate: 42P01
===============
```

5.8.10.3 SQLSTATE and SQLCODE

SQLSTATE is a five-character array. The five characters contain digits or upper-case letters that represent codes of various error and warning conditions. SQLSTATE has a hierarchical scheme: the first two characters indicate the general class of the condition, the last three characters indicate a subclass of the general condition. For example, the code 00000 indicates the success state.

SQLCODE is a simple integer. The value 0 indicates success, a positive value indicates success with additional information, a negative value indicates an error. The SQL standard only defines the positive value +100, which indicates that the last command returned or affected zero rows, and no specific negative values.

Table 5-91 Mapping between SQLSTATE and SQLCODE

SQLCODE Value	SQLSTATE Value	Description
0 (ECPG_NO_ERROR)	SQLSTATE 00000	Indicates no error.
100 (ECPG_NOT_FOUND)	SQLSTATE 02000	This is a harmless condition indicating that the last command retrieved or processed zero rows, or that you are at the end of the cursor. When processing a cursor in a loop, you could use this code as a way to detect when to abort the loop. An example is as follows: `while (1)` `{` `  EXEC SQL FETCH ... ;` `  if (sqlca.sqlcode == ECPG_NOT_FOUND)` `  break;` `}` Actually, WHENEVER NOT FOUND DO BREAK effectively does this internally, so there is usually no advantage in writing this out explicitly.
-12 (ECPG_OUT_OF_MEMORY)	SQLSTATE YE001	Indicates that your virtual memory is exhausted. The numeric value is defined as **–ENOMEM**.
-200 (ECPG_UNSUPPORTED)	SQLSTATE YE000	Indicates that the preprocessor has generated something that the library does not know about.

SQLCODE Value	SQLSTATE Value	Description
-201 (ECPG_TOO_MANY_ARGUMENTS)	SQLSTATE 07001 or 07002	Indicates that the command specified more host variables than the command expected.
-202 (ECPG_TOO_FEW_ARGUMENTS)	SQLSTATE 07001 or 07002	Indicates that the command specified fewer host variables than the command expected.
-203 (ECPG_TOO_MANY_MATCHES)	SQLSTATE 21000	Indicates that a query has returned multiple rows, but the statement is ready to store only one result row.
-204 (ECPG_INT_FORMAT)	SQLSTATE 42804	The host variable is of the int type and the data in the database is of a different type and contains a value that cannot be interpreted as an int. The library uses strtol() for this conversion.
-205 (ECPG_UINT_FORMAT)	SQLSTATE 42804	The host variable is of the unsigned int type and the data in the database is of a different type and contains a value that cannot be interpreted as an unsigned int. The library uses strtoul() for this conversion.
-206 (ECPG_FLOAT_FORMAT)	SQLSTATE 42804	The host variable is of the float type and the data in the database is of another type and contains a value that cannot be interpreted as a float value. The library uses strtod() for this conversion.
-207 (ECPG_NUMERIC_FORMAT)	SQLSTATE 42804	The host variable is of the numeric type and the data in the database is of another type and contains a value that cannot be interpreted as a numeric value.
-208 (ECPG_INTERVAL_FORMAT)	SQLSTATE 42804	The host variable is of the interval type and the data in the database is of another type and contains a value that cannot be interpreted as an interval value.
-209 (ECPG_DATE_FORMAT)	SQLSTATE 42804	The host variable is of the date type and the data in the database is of another type and contains a value that cannot be interpreted as a date value.
-210 (ECPG_TIMESTAMP_FORMAT)	SQLSTATE 42804	The host variable is of the timestamp type and the data in the database is of another type and contains a value that cannot be interpreted as a timestamp value.

SQLCODE Value	SQLSTATE Value	Description
-211 (ECPG_CONVERT_BOOL)	SQLSTATE 42804	The host variable is of the Boolean type, but the data in the database is neither 't' nor 'f'.
-212 (ECPG_EMPTY)	SQLSTATE YE000	The statement sent to the SQL server was empty. (This usually does not occur in an embedded SQL program, so it may point to an internal error.)
-213 (ECPG_MISSING_INDICATOR)	SQLSTATE 22002	A null value was returned and no null indicator variable was provided.
-214 (ECPG_NO_ARRAY)	SQLSTATE 42804	An ordinary variable was used in a place that requires an array.
-215 (ECPG_DATA_NOT_ARRAY)	SQLSTATE 42804	The database returned an ordinary variable in a place that requires array value.
-216 (ECPG_ARRAY_INSERT)	SQLSTATE 42804	The value cannot be inserted into the array.
-220 (ECPG_NO_CONN)	SQLSTATE 08003	The program tried to access a connection that does not exist.
-221 (ECPG_NOT_CONN)	SQLSTATE YE000	The program tried to access a connection that does exist but is not open. (This is an internal error.)
-230 (ECPG_INVALID_STMT)	SQLSTATE 26000	The statement you are trying to use has not been prepared.
-239 (ECPG_INFORMIX_DUPLICATE_KEY)	SQLSTATE 23505	Duplicate key error, violation of unique constraint.
-240 (ECPG_UNKNOWN_DESCRIPTOR)	SQLSTATE 33000	The specified descriptor was not found. The statement you are trying to use has not been prepared.
-241 (ECPG_INVALID_DESCRIPTOR_INDEX)	SQLSTATE 07009	The specified descriptor was out of range.
-242 (ECPG_UNKNOWN_DESCRIPTOR_ITEM)	SQLSTATE YE000	An invalid descriptor item was requested. (This is an internal error.)

SQLCODE Value	SQLSTATE Value	Description
-243 (ECPG_VAR_NOT_NUMERIC)	SQLSTATE 07006	During the execution of a dynamic statement, the database returned a numeric value and the host variable was not numeric.
-244 (ECPG_VAR_NOT_CHAR)	SQLSTATE 07006	During the execution of a dynamic statement, the database returned a non-numeric value and the host variable was numeric.
-284 (ECPG_INFORMIX_SUBSELECT_NOT_ONE)	SQLSTATE 21000	The result of the subquery was not a single row.
-400 (ECPG_PGSQL)	-	Some error caused by the SQL server. This message contains an error message from the SQL server.
-401 (ECPG_TRANS)	SQLSTATE 08007	The SQL server notified that the transaction cannot be started, committed, or rolled back.
-402 (ECPG_CONNECT)	SQLSTATE 08001	The connection attempt to the database did not succeed.
-403 (ECPG_DUPLICATE_KEY)	SQLSTATE 23505	Duplicate key error, violation of unique constraint.
-404 (ECPG_SUBSELECT_NOT_ONE)	SQLSTATE 21000	The result of the subquery was not a single row.
-602 (ECPG_WARNING_UNKNOWN_PORTAL)	SQLSTATE 34000	An invalid cursor name was specified.
-603 (ECPG_WARNING_IN_TRANSACTION)	SQLSTATE 25001	A transaction is in progress.
-604 (ECPG_WARNING_NO_TRANSACTION)	SQLSTATE 25P01	There is no active (in-progress) transaction.
-605 (ECPG_WARNING_PORTAL_EXISTS)	SQLSTATE 42P03	An existing cursor name was specified.

CAUTION

- The SQLSTATE codes 22002, 07001, 07002, 07006, 07009, 33000, 42601, 42804, 42P03, YE000 and YE001 are newly added to ecpg for embedded SQL statements. Other SQLSTATE codes are inherited from the kernel SQLSTATE codes.
- If the value of **SQLSCODE** is **–400**, ecpg detects that the kernel server returns an error. The error code of the kernel SQLSTATE is used.

5.8.11 Preprocessor Directives

This section describes the preprocessing instructions provided by ECPG. The preprocessing instructions are used to process program instructions for macro definition, file inclusion, and conditional compilation.

5.8.11.1 Including Files

To include an external file in an embedded SQL program, use the following statements:

```
EXEC SQL INCLUDE filename;
EXEC SQL INCLUDE <filename>;
EXEC SQL INCLUDE "filename";
```

NOTE

- ECPG searches for files in the following sequence:
  1. Current directory
  2. /usr/local/include
  3. GaussDB Kernel directory, which is defined at build time
  4. /usr/include
- When **EXEC SQL INCLUDE** *"filename"* is used, only the current directory is searched.
- In each directory, ECPG will first look for the file name as given, and if not found will append .h to the file name and try again (unless the specified file name already has that suffix).
- The file name is case sensitive.

5.8.11.2 Directives: ifdef, ifndef, else, elif, and endif

ecpg provides ifdef, ifndef, else, elif, and endif conditional compilation instructions. During preprocessing, different parts of the program are compiled based on different conditions. When using the program, you need to add the EXEC SQL prefix keyword.

The following is an example.

```
EXEC SQL ifndef TZVAR;
EXEC SQL SET TIMEZONE TO 'GMT';
EXEC SQL elif TZNAME;
EXEC SQL SET TIMEZONE TO TZNAME;
EXEC SQL else;
EXEC SQL SET TIMEZONE TO TZVAR;
EXEC SQL endif;
```


5.8.11.3 Directives define and undef

Similar to the directive **#define** that is known from C, embedded SQL has a similar concept.

```
EXEC SQL DEFINE name;
EXEC SQL DEFINE name value;
EXEC SQL UNDEF name;
```

The following is an example.

```
/* Define a name. */
EXEC SQL DEFINE HAVE_FEATURE;

/* Define constants. */
EXEC SQL DEFINE MYNUMBER 12;
EXEC SQL DEFINE MYSTRING 'abc';

/* Use undef to remove a previous definition. */
EXEC SQL UNDEF MYNUMBER;
```

You can also use the C versions **#define** and **#undef** in your embedded SQL program. The difference is where your defined values get evaluated. If you use **EXEC SQL DEFINE**, then ECPG evaluates the definitions and substitutes the values. In the following example, ECPG does the substitution and the compiler will never see any name or identifier **MYNUMBER**.

```
EXEC SQL DEFINE MYNUMBER 12;
...
EXEC SQL UPDATE Tbl SET col = MYNUMBER;
```

NOTICE

You cannot use **#define** for a variable used in an embedded SQL query because the embedded SQL precompiler cannot execute the declaration.

5.8.12 Using Library Functions

- ECPGdebug(int on, FILE *stream): If the first parameter of the function is not 0, the debug log function is enabled. The second parameter indicates the standard output stream of the log to be printed. Debug logs are executed on the standard output stream. The logs contain all input SQL statements and results from the GaussDB Kernel server.

  Example:

  ```
  #include <stdio.h>
  #include <stdlib.h>
  #include <string.h>
  #include "sqlca.h"

  int main()
  {
      ECPGdebug(1, stderr);
      /* Create testdb in advance. */
      EXEC SQL CONNECT TO testdb;
      EXEC SQL SET AUTOCOMMIT TO ON;
      EXEC SQL CREATE TABLE T1(a int);
      return (0);
  }
  ```

- ECPGget_PGconn(const char *_connection_name_): returns the database connection handle identified by the given name. If _connection_name_ is set to **NULL**, the current connection handle is returned. If no connection handle can be identified, the function returns **NULL**.

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "sqlca.h"

int main()
{
    ECPGdebug(1, stderr);
    /* Create testdb in advance. */
    EXEC SQL CONNECT TO testdb as con1;
    EXEC SQL SET AUTOCOMMIT TO ON;
    EXEC SQL DROP TABLE IF EXISTS T1;
    PGconn     *conn;
    conn = ECPGget_PGconn("con1");
    printf("conn = %p\n", conn);
    conn = ECPGget_PGconn(NULL);
    printf("conn = %p\n", conn);
    EXEC SQL CREATE TABLE T1(a int);
    return (0);
}
```

- ECPGtransactionStatus(const char * *connection_name*): returns the current transaction status of the *connection_name* connection. The possible return values are as follows:

  ```
  PQTRANS_IDLE,    /* connection idle */
  PQTRANS_ACTIVE,  /* command in progress */
  PQTRANS_INTRANS, /* idle, within transaction block */
  PQTRANS_INERROR, /* idle, within failed transaction */
  PQTRANS_UNKNOWN  /* cannot determine status */
  ```

  Example:

  ```
  #include <stdio.h>
  #include <stdlib.h>
  #include <string.h>
  #include "sqlca.h"

  int main()
  {
      ECPGdebug(1, stderr);
      /* Create testdb in advance. */
      EXEC SQL CONNECT TO testdb as con1;
      EXEC SQL DROP TABLE IF EXISTS T1;
      int a = ECPGtransactionStatus("con1");
      printf("%d\n", a);
      EXEC SQL CREATE TABLE T1(a int);
      EXEC SQL COMMIT;
      return (0);
  }
  ```

- ECPGfree_auto_mem(): releases all memory allocated for output host variables. This function is called (return\exit) when the program ends.

  Example:

  ```
  #include <stdio.h>
  #include <stdlib.h>
  #include <string.h>
  #include "sqlca.h"

  int main()
  {
    EXEC SQL BEGIN DECLARE SECTION;
    int *ip1=0;
    char **cp2=0;
    int *ipointer1=0;
    int *ipointer2=0;
    int column;
    EXEC SQL END DECLARE SECTION;
    int i;
  ```



```
  ECPGdebug(1, stderr);

  EXEC SQL WHENEVER SQLERROR DO sqlprint();
  EXEC SQL CONNECT TO REGRESSDB1;
  /* Create testdb in advance. */

  EXEC SQL CREATE TABLE test (a int, b text);
  EXEC SQL INSERT INTO test VALUES (1, 'one');
  EXEC SQL INSERT INTO test VALUES (2, 'two');
  EXEC SQL INSERT INTO test VALUES (NULL, 'three');
  EXEC SQL INSERT INTO test VALUES (NULL, NULL);

  EXEC SQL ALLOCATE DESCRIPTOR mydesc;
  EXEC SQL SELECT * INTO SQL DESCRIPTOR mydesc FROM test;
  EXEC SQL GET DESCRIPTOR mydesc :column=COUNT;
  EXEC SQL GET DESCRIPTOR mydesc VALUE 1 :ip1=DATA, :ipointer1=INDICATOR;
  EXEC SQL GET DESCRIPTOR mydesc VALUE 2 :cp2=DATA, :ipointer2=INDICATOR;

  printf("Result (%d columns):\n", column);
  for (i=0;i < sqlca.sqlerrd[2];++i)
  {
    if (ipointer1[i]) printf("NULL, ");
    else printf("%d, ",ip1[i]);

    if (ipointer2[i]) printf("NULL, ");
    else printf("'%s', ",cp2[i]);
    printf("\n");
  }
  ECPGfree_auto_mem();
  printf("\n");

  EXEC SQL DEALLOCATE DESCRIPTOR mydesc;
  EXEC SQL ROLLBACK;
  EXEC SQL DISCONNECT;
  return 0;
}
```

5.8.13 SQL Descriptor Area

An SQL descriptor area (SQLDA) is a more sophisticated method for processing the result of a SELECT, FETCH or a DESCRIBE statement. An SQLDA groups the data of one row of data together with metadata items into one data structure. ecpg provides two ways to use descriptor areas: named SQLDA and C-structure SQLDA.

5.8.13.1 Named SQLDA

A named SQLDA consists of a header and one or more item descriptor areas, which basically each describe one column in the result row. The header contains information concerning the entire descriptor area, and each item descriptor area describes a column in the result row.

- Before using an SQLDA, you need to allocate it.
  ```
  EXEC SQL ALLOCATE DESCRIPTOR identifier;
  ```
- When you no longer need the SQLDA, deallocate it in time.
  ```
  EXEC SQL DEALLOCATE DESCRIPTOR identifier;
  ```
- To use a descriptor area, declare it using the INTO clause.
  ```
  EXEC SQL FETCH NEXT FROM mycursor INTO SQL DESCRIPTOR mydesc;
  ```
  If the result set is empty, the descriptor area still contains the metadata from the query.
- For a prepared query that has not been executed, you can use DESCRIBE to obtain the metadata of its result set.


```
EXEC SQL BEGIN DECLARE SECTION;
   char *sql_stmt = "SELECT * FROM table1";
EXEC SQL END DECLARE SECTION;

   EXEC SQL PREPARE stmt1 FROM :sql_stmt;
   EXEC SQL DESCRIBE stmt1 INTO SQL DESCRIPTOR mydesc;
```

In DESCRIBE and FETCH statements, the INTO and USING keywords are used similarly: they produce a result set and metadata in a descriptor area.

- To retrieve the value in a descriptor area from the header and store it into a host variable, use the following command:
  ```
  EXEC SQL GET DESCRIPTOR name :hostvar = field;
  ```
- Currently, only one header descriptor area **COUNT** is defined, which tells how many item descriptor areas exist (that is, how many columns are contained in the result). The host variable must be of the integer type. To retrieve a specific value from the item descriptor area, run the following command:
  ```
  EXEC SQL GET DESCRIPTOR name VALUE num :hostvar = field;
  ```

  num can be a character integer or a host variable that contains an integer. Possible data types are as follows:

  - CARDINALITY (integer): number of rows in the result set
  - DATA: actual data item (therefore, the data type of this field depends on the query)
  - DATETIME_INTERVAL_CODE (integer): When **TYPE** is **9**, **DATETIME_INTERVAL_CODE** will have a value of **1** for DATE, **2** for TIME, **3** for TIMESTAMP, **4** for TIME WITH TIME ZONE, or **5** for TIMESTAMP WITH TIME ZONE.
  - INDICATOR (integer): indicator (indicating a null value or a value truncation)
  - LENGTH (integer): data length in characters
  - NAME(string): column name
  - OCTET_LENGTH (integer): length of the character representation of the data in bytes
  - PRECISION (integer): precision (for the numeric type)
  - RETURNED_LENGTH (integer): data length in characters
  - RETURNED_OCTET_LENGTH (integer): length of the character representation of the data in bytes
  - SCALE (integer): ratio (for the numeric type)
  - TYPE (integer): numeric code of the data type of the column
- Retrieve the column value and store it in a host variable.
  ```
  EXEC SQL GET DESCRIPTOR mydesc  VALUE num :hostvar = field
  ```

  num can be a character integer or a host variable that contains an integer. Possible fields are as follows:

  - DATA
  - Actual data item (the data type of this column depends on the query)
  - NAME(string)
  - Field name
- To manually create a descriptor area to provide input parameters for a query or cursor, run the following command:
  ```
  EXEC SQL SET DESCRIPTOR name VALUE numfield = :hostvar;
  ```


- To retrieve multiple rows of records in a FETCH statement and use a host variable of the array type to store data, run the following commands:

```
EXEC SQL BEGIN DECLARE SECTION;
   int id[5];
EXEC SQL END DECLARE SECTION;
   EXEC SQL FETCH 5 FROM mycursor INTO SQL DESCRIPTOR mydesc;
   EXEC SQL GET DESCRIPTOR mydesc VALUE 1 :id = DATA;
```

5.8.13.2 C-Structure SQLDA

SQLDA is a C language structure used to store a query result set. A structure stores a record of a result set.

```
EXEC SQL include sqlda.h;
sqlda_t      *MySQLda;
EXEC SQL FETCH 3 FROM mycursor INTO DESCRIPTOR MySQLda;
```

Note that the SQL keyword is omitted. The paragraphs about the use cases of the INTO and USING keywords in section **Named SQLDA** also apply here. In a DESCRIBE statement, if the INTO keyword is used, the DESCRIPTOR keyword can be omitted.

```
EXEC SQL DESCRIBE prepared_statement INTO MySQLda;
```

- Procedure
  a. Prepare a query and declare a cursor for it.
  b. Declare an SQLDA for the result row.
  c. Declare SQLDA for input parameters, initialize parameters, and allocate memory.
  d. Open a cursor with the input SQLDA.
  e. Fetch rows from the cursor and store them in the output SQLDA.
  f. Read the value from the output SQLDA to the host variable.
  g. Close the cursor.
  h. Deallocate the memory allocated to the SQLDA.
- There are three types of SQLDA data structures: sqlda_t, sqlvar_t, and struct sqlname.
  a. sqlda_t

     The definition of sqlda_t is as follows:

```
struct sqlda_struct
{
   char        sqldaid[8];
   long        sqldabc;
   short       sqln;
   short       sqld;
   struct sqlda_struct *desc_next;
   struct sqlvar_struct sqlvar[1];
};
typedef struct sqlda_struct sqlda_t;
```

     The structure members are described as follows:

     - **sqldaid**: contains a string "SQLDA".
     - **sqldabc**: contains the size (in bytes) of the allocated space.
     - **sqln**: contains the number of input parameters for a parameterized query in case it is passed into OPEN, DECLARE or EXECUTE statements using the USING keyword. When it is used as the output

of a SELECT, EXECUTE, or FETCH statement, its value is the same as that of **sqld**.

- **sqld**: contains the number of fields in a result set.
- **desc_next**: If the query returns more than one record, multiple linked SQLDA structures are returned, and **desc_next** holds a pointer to the next SQLDA structure in the list.
- **sqlvar**: indicates the array of the columns in the result set.

b. sqlvar_t

The structure type sqlvar_t holds a column value and metadata (such as type and length). The definition of this type is as follows:

```
struct sqlvar_struct
{
    short          sqltype;
    short          sqllen;
    char          *sqldata;
    short         *sqlind;
    struct sqlname sqlname;
};
typedef struct sqlvar_struct sqlvar_t;
```

The structure members are described as follows:

- **sqltype**: contains the type identifier of the field.
- **sqllen:** contains the binary length of the field, for example, 4 bytes for ECPGt_int.
- **sqldata**: points to the data. For details about the data format, see **Type Mapping**.
- **sqlind**: points to a null indicator. The value **0** indicates not null, and the value **–1** indicates null.
- **sqlname**: indicates the name of the field.

c. struct sqlname

A struct sqlname structure holds a column name. It is treated as a member of the sqlvar_t structure. The definition of this type is as follows:

```
#define NAMEDATALEN 64
struct sqlname
{
    short          length;
    char           data[NAMEDATALEN];
};
```

The structure members are described as follows:

- **length**: contains the length of the field name.
- **data**: contains the actual field name.

- Use an SQLDA to retrieve a result set.

  The general procedure for retrieving a query result set through an SQLDA is as follows:

  a. Declare an sqlda_t structure to receive the result set.
  b. Execute the **FETCH**, **EXECUTE**, or **DESCRIBE** command to process a query for which an SQLDA has been declared.

c. Check the number of records in the result set by looking at sqln, a member of the sqlda_t structure.

d. Fetch the values of each column from sqlvar[0], sqlvar[1], ..., members of the sqlda_t structure.

e. Go to next row (sqlda_t) by following the desc_next pointer, a member of the sqlda_t structure.

f. Repeat the preceding steps as required.

For example:

```
/* Declare an sqlda_t structure to receive the result set. */
sqlda_t *sqlda1;
/* Next, specify an SQLDA in a command. This is an example of the FETCH command. */
EXEC SQL FETCH NEXT FROM cur1 INTO DESCRIPTOR sqlda1;
/* Run a loop to retrieve rows along the linked list. */
sqlda_t *cur_sqlda;
for (cur_sqlda = sqlda1;
     cur_sqlda != NULL;
     cur_sqlda = cur_sqlda->desc_next)
{
    ...
}
/* Inside the loop, run another loop to retrieve the data of each column in the row (sqlvar_t). */
for (i = 0; i < cur_sqlda->sqld; i++)
{
    sqlvar_t v = cur_sqlda->sqlvar[i];
    char *sqldata = v.sqldata;
    short sqllen  = v.sqllen;
    ...
}
/* To fetch the values of a column, check the value of the sqltype member of the sqlvar_t structure.
Then, switch to an appropriate method based on the column type to copy data from the sqlvar field
to a host variable. */
char var_buf[1024];
switch (v.sqltype)
{
    case ECPGt_char:
        memset(&var_buf, 0, sizeof(var_buf));
        memcpy(&var_buf, sqldata, (sizeof(var_buf) <= sqllen ? sizeof(var_buf) - 1 : sqllen));
        break;

    case ECPGt_int:
        memcpy(&intval, sqldata, sqllen);
        snprintf(var_buf, sizeof(var_buf), "%d", intval);
        break;
    ...
}
```

- Use an SQLDA to pass query parameters.

  The general procedure for passing input parameters to a prepared query using an SQLDA is as follows:

  a. Create a prepared query (prepared statement).

  b. Declare an sqlda_t structure as an SQLDA.

  c. Allocate a memory area for the SQLDA.

  d. Set (copy) the input values in the allocated memory.

  e. Open a cursor declaring the SQLDA.

  For example:

```
/* First, create a prepared statement. */
EXEC SQL BEGIN DECLARE SECTION;
    char query[1024] = "SELECT d.oid, * FROM pg_database d, pg_stat_database s WHERE d.oid =
s.datid AND (d.datname = ? OR d.oid = ?)";
EXEC SQL END DECLARE SECTION;
```



```
    EXEC SQL PREPARE stmt1 FROM :query;

    /* Allocate memory for an SQLDA and set the number of input parameters in the sqln member variable of the sqlda_t structure.
    * When the prepared query requires two or more input parameters, the application must allocate extra memory space. The space size is calculated as follows: (Number of parameters – 1) x sizeof(sqlvar_t).
    * The example here shows how to allocate memory space for two input parameters.
     */
    sqlda_t *sqlda2;
    sqlda2 = (sqlda_t *) malloc(sizeof(sqlda_t) + sizeof(sqlvar_t));
    memset(sqlda2, 0, sizeof(sqlda_t) + sizeof(sqlvar_t));
    sqlda2->sqln = 2; /* Number of input variables */
    /* After memory allocation, store the parameter values into the sqlvar[] array. (This is same array used for retrieving column values when the SQLDA is receiving a result set.)
    * In this example, the input parameters are testdb (string type) and 1 (integer type). */
    sqlda2->sqlvar[0].sqltype = ECPGt_char;
    sqlda2->sqlvar[0].sqldata = "testdb";
    sqlda2->sqlvar[0].sqllen  = 8;
    int intval = 1;
    sqlda2->sqlvar[1].sqltype = ECPGt_int;
    sqlda2->sqlvar[1].sqldata = (char *) &intval;
    sqlda2->sqlvar[1].sqllen  = sizeof(intval);
    /* Input parameters are passed to the prepared statement by opening a cursor and declaring the SQLDA that has been created. */
    EXEC SQL OPEN cur1 USING DESCRIPTOR sqlda2;
    /* Finally, the allocated memory must be explicitly released after you use the input SQLDA, which is different from the SQLDA used to receive query results. */
    free(sqlda2);
```

5.8.14 Examples

ECPG Example Code

```
#include <locale.h>
#include <string.h>
#include <stdlib.h>

exec sql whenever sqlerror sqlprint;
exec sql include sqlca;

int main(void)
{
EXEC SQL BEGIN DECLARE SECTION;
    char *temp_str = (char *)malloc(11);
EXEC SQL END DECLARE SECTION;

    ECPGdebug(1, stderr);
    /* Create the testdb database in advance. */
    exec sql connect to testdb;

    /* Enable the automatic commit function. You do not need to manually commit the exec sql command. */
    exec sql set autocommit = on;
    exec sql drop table if exists test_t;
    /* Create a table and insert data. */
    exec sql create table test_t(f float, i int, a int[10], mstr char(10));
    exec sql insert into test_t(f, i, a, mstr) values(1.01,1,'{0,1,2,3,4,5,6,7,8,9}', 'China');

    /* Disable the automatic commit function. The following SQL statements for inserting data must be manually committed: */
    exec sql set autocommit = off;
    exec sql insert into test_t(f, i, a, mstr) values(2.01,2,'{0,1,2,3,4,5,6,7,8,9}', 'USA');
    exec sql commit;

    exec sql insert into test_t(f, i, a, mstr) values(3.01,3,'{0,1,2,3,4,5,6,7,8,9}', 'AUS');
    exec sql insert into test_t(f, i, a, mstr) values(4.01,4,'{0,1,2,3,4,5,6,7,8,9}', 'JAP');
    exec sql commit;
```



```
EXEC SQL BEGIN DECLARE SECTION;
    int a[10] = {9,8,7,6,5,4,3,2,1,0};
    int id = 6;
EXEC SQL END DECLARE SECTION;

    /* Fetch data from the host variable and insert the data into the table. The type of the host variable is
the same as that defined in the table. */
    strcpy(temp_str, "RUS");
    exec sql insert into test_t(f, i, a, mstr) values(5.01,5,:a,:temp_str);
    exec sql commit;

    exec sql set autocommit = on;
    exec sql begin;
    exec sql insert into test_t(f, i, a, mstr) values(6.01,:id,:a,'SIG');
    exec sql commit;
    exec sql set autocommit = off;

exec sql begin declare section;
    float ff;
    char tmp_text[25] = "klmnopqrst";
exec sql end declare section;

    exec sql set autocommit = on;
    exec sql begin work;

    printf("Found ff=%f tmp_text=%10.10s\n", ff, tmp_text);

    /* Example of a conditional query statement */
    exec sql select f, mstr into :ff,:tmp_text from test_t where f > (select f from test_t where i = 4 or i < 0)
order by a limit 1;
    printf("Found ff=%f tmp_text=%10.10s\n", ff, tmp_text);

    exec sql select f, mstr into :ff,:tmp_text from test_t where mstr = 'JAP' order by i;
    printf("Found ff=%f tmp_text=%10.10s\n", ff, tmp_text);

    exec sql select f, mstr into :ff,:tmp_text from  test_t order by i DESC limit 1;
    printf("Found ff=%f tmp_text=%10.10s\n", ff, tmp_text);

    exec sql select f, mstr into :ff,:tmp_text from  test_t order by mstr limit 1;
    printf("Found ff=%f tmp_text=%10.10s\n", ff, tmp_text);

    exec sql select count(f), a into :ff,:tmp_text from test_t where i > 2 group by a limit 1;
    printf("Found ff=%f tmp_text=%20.30s\n", ff, tmp_text);

    exec sql select count(f), a into :ff,:tmp_text from test_t where i > 3 group by a order by a limit 1;
    printf("Found ff=%f tmp_text=%20.30s\n", ff, tmp_text);

    exec sql select sum(f), a into :ff,:tmp_text from test_t where i > 2  group by a order by a limit 1;
    printf("Found ff=%f tmp_text=%20.30s\n", ff, tmp_text);

    exec sql select distinct a into :tmp_text from test_t order by a limit 1;

    exec sql drop table test_t;

    exec sql commit;
    /* Release the connection and release the memory allocated to the host variable. */
    exec sql disconnect;
    free(temp_str);

    return 0;
}
```

Example Code of the pgtypes Library Function

Example 1: Use library functions to perform different operations on time and date types. For details, see **Using Library Functions**.

```
#include <stdio.h>
#include <string.h>
```



```
#include <stdlib.h>
#include <limits.h>
#include <pgtypes_date.h>
#include <pgtypes_timestamp.h>

char *dates[] = { "19990108foobar",
                  "19990108 foobar",
                  "1999-01-08 foobar",
                  "January 8, 1999",
                  "1999-01-08",
                  "1/8/1999",
                  "1/18/1999",
                  "01/02/03",
                  "1999-Jan-08",
                  "Jan-08-1999",
                  "08-Jan-1999",
                  "99-Jan-08",
                  "08-Jan-99",
                  "08-Jan-06",
                  "Jan-08-99",
                  "19990108",
                  "990108",
                  "1999.008",
                  "J2451187",
                  "January 8, 99 BC",
                  NULL
                 };

/* The value cannot conflict with the value of times of libc. */
static char *times[] = { "0:04",
                         "1:59 PDT",
                         "13:24:40 -8:00",
                         "13:24:40.495+3",
                          NULL
                        };

char *intervals[] = { "1 minute",
                      "1 12:59:10",
                      "2 day 12 hour 59 minute 10 second",
                      "1 days 12 hrs 59 mins 10 secs",
                      "1 days 1 hours 1 minutes 1 seconds",
                      "1 year 59 mins",
                      "1 year 59 mins foobar",
                      NULL
                    };

int main(void)
{
 exec sql begin declare section;
    date date1;
    timestamp ts1, ts2;
    char *text;
    interval *i1;
    date *dc;
 exec sql end declare section;

    int i, j;
    char *endptr;

    ECPGdebug(1, stderr);

    /* Parse a timestamp from its textual representation and convert the date into a character string. */
    ts1 = PGTYPEStimestamp_from_asc("2003-12-04 17:34:29", NULL);
    text = PGTYPEStimestamp_to_asc(ts1);

    printf("timestamp: %s\n", text);
    free(text);

    /* Extract the date part from the timestamp. */
```



```
    date1 = PGTYPESdate_from_timestamp(ts1);
    dc = PGTYPESdate_new();
    *dc = date1;
    /* Return the textual representation of a date variable. */
    text = PGTYPESdate_to_asc(*dc);
    printf("Date of timestamp: %s\n", text);
    free(text);
    PGTYPESdate_free(dc);

    for (i = 0; dates[i]; i++)
    {
        bool err = false;
        /* Parse a date from the textual representation of the date. */
        date1 = PGTYPESdate_from_asc(dates[i], &endptr);
        if (date1 == INT_MIN) {
            err = true;
        }
        /* Return the textual representation of a date variable. */
        text = PGTYPESdate_to_asc(date1);
        printf("Date[%d]: %s (%c - %c)\n",
                i, err ? "-" : text,
                endptr ? 'N' : 'Y',
                err ? 'T' : 'F');
        free(text);
        if (!err)
        {
            for (j = 0; times[j]; j++)
            {
                int length = strlen(dates[i])+ 1+ strlen(times[j])+ 1;
                char* t = (char *)malloc(length);
                sprintf(t, "%s %s", dates[i], times[j]);
                /* Parse a timestamp from its textual representation and convert the date into a character string.
*/
                ts1 = PGTYPEStimestamp_from_asc(t, NULL);
                text = PGTYPEStimestamp_to_asc(ts1);
                if (i != 19 || j != 3)
                    printf("TS[%d,%d]: %s\n",i, j, errno ? "-" : text);
                free(text);
                free(t);
            }
        }
    }

    /* Parse a timestamp from its textual representation. */
    ts1 = PGTYPEStimestamp_from_asc("2004-04-04 23:23:23", NULL);

    for (i = 0; intervals[i]; i++)
    {
        interval *ic;
        /* Parse an interval from its textual representation. */
        i1 = PGTYPESinterval_from_asc(intervals[i], &endptr);
        if (*endptr)
            printf("endptr set to %s\n", endptr);
        if (!i1)
        {
             printf("Error parsing interval %d\n", i);
             continue;
         }
        /* Add an interval variable to the timestamp variable. */
         j = PGTYPEStimestamp_add_interval(&ts1, i1, &ts2);
         if (j < 0)
             continue;
        /* Convert a variable of the interval type to the text format. */
         text = PGTYPESinterval_to_asc(i1);
         printf("interval[%d]: %s\n", i, text ? text : "-");
         free(text);

        /* Return a pointer to an allocated interval variable. */
         ic = PGTYPESinterval_new();
```



```
        /*Copy a variable of the interval type.*/
        PGTYPESinterval_copy(i1, ic);
        /* Convert a variable of the interval type to the text format. */
        text = PGTYPESinterval_to_asc(i1);
        printf("interval_copy[%d]: %s\n", i, text ? text : "-");
        free(text);
        /* Release the memory that has been allocated to an interval variable. */
        PGTYPESinterval_free(ic);
        PGTYPESinterval_free(i1);
    }

    return (0);
}
```

Example 2: Use the pgtypes library function to perform different operations on the numeric type.

```
#include <stdio.h>
#include <stdlib.h>
#include <pgtypes_numeric.h>
#include <pgtypes_error.h>
#include <decimal.h>

char* nums[] = { "2E394", "-2", ".794", "3.44", "592.49E21", "-32.84e4",
                 "2E-394", ".1E-2", "+.0", "-592.49E-07", "+32.84e-4",
                 ".500001", "-.5000001",
                 "12345678901234567890123456789.91", /* A 30-bit number should be converted into a decimal number.*/
                 "12345678901234567890123456789.921", /* A 31-bit number cannot be converted into a decimal number. */
                 "not a number",
                 NULL
                };

static void check_errno(void);

int main(void)
{
    char *text="error\n";
    char *endptr;
    numeric *num, *nin;
    decimal *dec;
    long l;
    int i, j, k, q, r, count = 0;
    double d;
    numeric **numarr = (numeric **) calloc(1, sizeof(numeric));

    ECPGdebug(1, stderr);

    for (i = 0; nums[i]; i++)
    {
        /* Return a pointer to a string allocated by malloc that contains the string representation of the numeric type nums[i]. */
        num = PGTYPESnumeric_from_asc(nums[i], &endptr);
        if (!num) check_errno();
        if (endptr != NULL)
        {
            printf("endptr of %d is not NULL\n", i);
            if (*endptr != '\0')
                printf("*endptr of %d is not \\0\n", i);
        }
        if (!num) continue;

        numarr = (numeric **)realloc(numarr, sizeof(numeric *) * (count + 1));
        numarr[count++] = num;

        /* Return a pointer to a string allocated by malloc that contains the string representation of the numeric type num. */
        text = PGTYPESnumeric_to_asc(num, -1);
        if (!text) check_errno();
```



```
        printf("num[%d,1]: %s\n", i, text); free(text);
        text = PGTYPESnumeric_to_asc(num, 0);
        if (!text) check_errno();
         printf("num[%d,2]: %s\n", i, text); free(text);
        text = PGTYPESnumeric_to_asc(num, 1);
        if (!text) check_errno();
         printf("num[%d,3]: %s\n", i, text); free(text);
        text = PGTYPESnumeric_to_asc(num, 2);
        if (!text) check_errno();
        printf("num[%d,4]: %s\n", i, text); free(text);

        /* Request a pointer to a newly allocated numeric variable. */
        nin = PGTYPESnumeric_new();
        text = PGTYPESnumeric_to_asc(nin, 2);
        if (!text) check_errno();
        printf("num[%d,5]: %s\n", i, text); free(text);

        /* Convert a numeric variable into a long int variable. */
        r = PGTYPESnumeric_to_long(num, &l);
        if (r) check_errno();
        printf("num[%d,6]: %ld (r: %d)\n", i, r?0L:l, r);
        if (r == 0)
        {
          /* Convert a long int variable into a numeric variable. */
           r = PGTYPESnumeric_from_long(l, nin);
           if (r) check_errno();
       /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type nin. */
           text = PGTYPESnumeric_to_asc(nin, 2);
           /* Compare two numeric variables. */
           q = PGTYPESnumeric_cmp(num, nin);
           printf("num[%d,7]: %s (r: %d - cmp: %d)\n", i, text, r, q);
           free(text);
         }

         /* Convert a numeric variable into an int variable. */
         r = PGTYPESnumeric_to_int(num, &k);
         if (r) check_errno();
         printf("num[%d,8]: %d (r: %d)\n", i, r?0:k, r);
         if (r == 0)
         {
            /* Convert an int variable into a numeric variable. */
            r = PGTYPESnumeric_from_int(k, nin);
            if (r) check_errno();
       /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type nin. */
            text = PGTYPESnumeric_to_asc(nin, 2);
            q = PGTYPESnumeric_cmp(num, nin);
            printf("num[%d,9]: %s (r: %d - cmp: %d)\n", i, text, r, q);
            free(text);
         }

         if (i != 6)
         {
             /* Convert a variable of the numeric type to the double-precision type. */
             r = PGTYPESnumeric_to_double(num, &d);
             if (r) check_errno();
             printf("num[%d,10]: %g (r: %d)\n", i, r?0.0:d, r);
          }

         /* Request a pointer to a newly allocated numeric variable. */
         dec = PGTYPESdecimal_new();
         /* Convert a decimal variable into a numeric variable. */
         r = PGTYPESnumeric_to_decimal(num, dec);
         if (r) check_errno();
         printf("num[%d,11]: - (r: %d)\n", i, r);
         if (r == 0)
         {
           /* Convert a decimal variable into a numeric variable. */
```



```
            r = PGTYPESnumeric_from_decimal(dec, nin);
            if (r) check_errno();
        /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type nin. */
            text = PGTYPESnumeric_to_asc(nin, 2);
           /* Compare two numeric variables. */
            q = PGTYPESnumeric_cmp(num, nin);
            printf("num[%d,12]: %s (r: %d - cmp: %d)\n", i, text, r, q);
            free(text);
        }

        /* Release the memory allocated to the numeric variable. */
        PGTYPESdecimal_free(dec);
        PGTYPESnumeric_free(nin);
        printf("\n");
    }

    for (i = 0; i < count; i++)
    {
        for (j = 0; j < count; j++)
        {
          /* Request a pointer to a newly allocated numeric variable. */
          numeric* a = PGTYPESnumeric_new();
          numeric* s = PGTYPESnumeric_new();
          numeric* m = PGTYPESnumeric_new();
          numeric* d = PGTYPESnumeric_new();
          /* Add two numeric variables to the third numeric variable. */
          r = PGTYPESnumeric_add(numarr[i], numarr[j], a);
          if (r)
          {
             check_errno();
             printf("r: %d\n", r);
           }
          else
          {
            /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type a. */
              text = PGTYPESnumeric_to_asc(a, 10);
              printf("num[a,%d,%d]: %s\n", i, j, text);
              free(text);
          }
          /* Subtract two numeric variables and returns the result to the third numeric variable. */
          r = PGTYPESnumeric_sub(numarr[i], numarr[j], s);
          if (r)
          {
              check_errno();
              printf("r: %d\n", r);
          }
          else
          {
            /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type s. */
              text = PGTYPESnumeric_to_asc(s, 10);
              printf("num[s,%d,%d]: %s\n", i, j, text);
              free(text);
          }
          /* Multiply two numeric variables and returns the result to the third numeric variable. */
          r = PGTYPESnumeric_mul(numarr[i], numarr[j], m);
          if (r)
          {
              check_errno();
              printf("r: %d\n", r);
          }
          else
          {
            /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type m. */
              text = PGTYPESnumeric_to_asc(m, 10);
              printf("num[m,%d,%d]: %s\n", i, j, text);
```



```
            free(text);
        }
        /* Divide two numeric variables and returns the result to the third numeric variable. */
        r = PGTYPESnumeric_div(numarr[i], numarr[j], d);
        if (r)
        {
            check_errno();
            printf("r: %d\n", r);
        }
        else
        {
          /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type d. */
            text = PGTYPESnumeric_to_asc(d, 10);
            printf("num[d,%d,%d]: %s\n", i, j, text);
            free(text);
        }

       /* Release the memory allocated to the numeric variable. */
        PGTYPESnumeric_free(a);
        PGTYPESnumeric_free(s);
        PGTYPESnumeric_free(m);
        PGTYPESnumeric_free(d);
    }
  }

  for (i = 0; i < count; i++)
  {
      /* Return a pointer to a string allocated by malloc that contains the string representation of the
numeric type numarr[i]. */
      text = PGTYPESnumeric_to_asc(numarr[i], -1);
      printf("%d: %s\n", i, text);
      free(text);
      /* Free memory. */
      PGTYPESnumeric_free(numarr[i]);
  }
  free(numarr);

  return (0);
}

/* Error handling... */
static void
check_errno(void)
{
  switch(errno)
  {
      case 0:
      printf("(no errno set) - ");
      break;
      case PGTYPES_NUM_OVERFLOW:
      printf("(errno == PGTYPES_NUM_OVERFLOW) - ");
      break;
      case PGTYPES_NUM_UNDERFLOW:
      printf("(errno == PGTYPES_NUM_UNDERFLOW) - ");
      break;
      case PGTYPES_NUM_BAD_NUMERIC:
      printf("(errno == PGTYPES_NUM_BAD_NUMERIC) - ");
      break;
      case PGTYPES_NUM_DIVIDE_ZERO:
      printf("(errno == PGTYPES_NUM_DIVIDE_ZERO) - ");
      break;
      default:
      printf("(unknown errno (%d))\n", errno);
      printf("(libc: (%s)) ", strerror(errno));
      break;
  }
}
```


5.8.15 ecpg and Pro*C Compatibility Comparison

ecpg, provided by GaussDB, is an embedded SQL preprocessor for C programs. It differs from the Pro*C preprocessor of the A-compatible database in the behavior and semantics of compiling and executing commands, syntax, and embedded statements.

Comparison between ecpg and Pro*C:

- Currently, ecpg does not support EXEC SQL CONTEXT ALLOCATE, EXEC SQL CONTEXT USE, and EXEC SQL Context FREE.

  NOTE

  Currently, ecpg does not support context allocation, use, and release. ecpg has an independent memory management mechanism. In multi-thread mode, ecpg independently establishes connections, executes SQL statements, and releases related resources in each thread. This usage mode is the same as the processing logic for each thread to allocate and release contexts in Pro*C multi-thread mode.

- Currently, ecpg does not support EXEC SQL COMMIT WORK RELEASE.

  NOTE

  In ecpg, after the COMMIT statement is executed, the **RELEASE** option is not available. You need to execute the EXEC SQL DISCONNECT and EXEC SQL CLOSE statements to release related resources. In Pro*C, EXEC SQL COMMIT has the **RELEASE** option, which is used to release all resource information such as connections and cursors held by the program.

- Currently, ecpg does not support EXEC SQL ENABLE THREAD.

  NOTE

  To enable macro definition in the ecpg compilation options, you need to define ENABLE_THREAD_SAFETY in the .pgc file of the main function.

- Currently, ecpg does not support features or syntaxes such as stored procedures, packages, anonymous blocks, and flashback.

5.8.16 ECPG API Reference

The ECPG API reference describes the data type-related interfaces provided by the pgtypes library for users to use in embedded SQL-C programs. The pgtypes library maps SQL data types to C data types and provides some interfaces to implement basic functions and calculations.

5.8.16.1 Interval Type

Table 5-92 lists the common APIs for interval data provided by the ECPG.

Table 5-92 Common interval type APIs

API	Description	Remarks
interval* PGTYPESinterval_new(void)	Returns a pointer to an allocated interval variable.	This function creates an interval variable on the heap. The return value is of the interval* type.

API	Description	Remarks
void PGTYPESinterval_free(interval* inval)	Releases the memory that has been allocated to an interval variable.	This function releases the memory allocated to the interval* variable created by the PGTYPESinterval_new function.
interval* PGTYPESinterval_from_asc(char* str, char** endptr)	Parses an interval from its textual representation.	This function parses the input string **str** and returns a pointer to an allocated interval variable. Currently, ECPG parses the entire character string and does not support storing the address of the first invalid character in ***endptr**. You can set **endptr** to **NULL**.
char* PGTYPESinterval_to_asc(interval* span)	Converts a variable of the interval type into its textual representation.	This function converts the interval variable to which the span points into a char*.
int PGTYPESinterval_copy(interval* intvlsrc, interval* intvldest)	Copies a variable of the interval type.	This function copies the interval variable that intvlsrc points to into the variable that intvldest points to.

Examples

For details, see **Examples**.

5.8.16.2 Numeric Types

Table 5-93 lists the common APIs for numeric (numeric or decimal) data provided by the ECPG.

Table 5-93 Common numeric type APIs

API	Description	Remarks
numeric* PGTYPESnumeric_new(void)	Requests a pointer to a newly allocated numeric variable.	This function creates a numeric variable on the heap. The return value is of the numeric* type.

API	Description	Remarks
decimal* PGTYPESdecimal_new(void)	Requests a pointer to a newly allocated decimal variable.	This function creates a decimal variable on the heap. The return value is of the decimal* type.
void PGTYPESnumeric_free(numeric* var)	Frees the memory of a variable of the numeric type.	This function frees a numeric* variable created by using the PGTYPESnumeric_new function.
void PGTYPESdecimal_free(decimal*)	Frees the memory of a variable of the decimal type.	This function frees a decimal* variable created by using the PGTYPESdecimal_new function.
numeric* PGTYPESnumeric_from_asc(char* str, char** endptr)	Converts a string to the numeric type.	For example, the valid formats are –2, .794, +3.44, 592.49E07 or –32.84e–4. If the value could be parsed successfully, a valid pointer is returned; otherwise, a null pointer is returned. ECPG only parses complete character strings. Currently, ECPG does not support storing the address of the first invalid character in ***endptr**. However, **endptr** can be set to null.
char* PGTYPESnumeric_to_asc(numeric* num, int dscale)	Returns a pointer to a string allocated by malloc that contains the string representation of the numeric type num.	The numeric value will be printed using dscale decimal places and will be rounded if necessary.
int PGTYPESnumeric_add(numeric* var1, numeric* var2, numeric* result)	Adds two numeric variables to the third numeric variable.	This function adds the variables *var1* and *var2* to the result variable *result*. The function returns **0** on success and **–1** in case of error.

API	Description	Remarks
int PGTYPESnumeric_sub(numeric* var1, numeric* var2, numeric* result)	Subtracts two numeric variables and returns the result to the third numeric variable.	This function subtracts *var2* from *var1*. The result of this operation is stored in the variable *result*. The function returns **0** on success and **–1** in case of error.
int PGTYPESnumeric_mul(numeric* var1, numeric* var2, numeric* result)	Multiplies two numeric variables and returns the result to the third numeric variable.	This function multiplies variables *var1* and *var2*. The result of this operation is stored in the variable *result*. The function returns **0** on success and **–1** in case of error.
int PGTYPESnumeric_div(numeric* var1, numeric* var2, numeric* result)	Divides two numeric variables and returns the result to the third numeric variable.	This function divides variable *var1* by variable *var2*. The result of this operation is stored in the variable *result*. The function returns **0** on success and **–1** in case of error.
int PGTYPESnumeric_cmp(numeric* var1, numeric* var2)	Compares two numeric variables.	This function compares two numeric variables. If an error occurs, *INT_MAX* is returned. On success, this function returns one of the following three possible results: ● If *var1* is greater than *var2*, **1** is returned. ● If *var1* is smaller than *var2*, **–1** is returned. ● If *var1* and *var2* are equal, **0** is returned.
int PGTYPESnumeric_from_int(signed int int_val, numeric* var)	Converts an int variable into a numeric variable.	This function accepts a signed int variable and stores it in the numeric variable *var*. The function returns **0** on success and **–1** in case of failure.

API	Description	Remarks
int PGTYPESnumeric_from_long(signed long int long_val, numeric* var)	Converts a long int variable into a numeric variable.	This function accepts a long int variable and stores it in the numeric variable *var*. The function returns **0** on success and **–1** in case of failure.
int PGTYPESnumeric_copy(numeric* src, numeric* dst)	Copies a numeric variable to another one.	This function copies the value of the variable that **src** points to into the variable that **dst** points to. The function returns **0** on success and **–1** in case of error.
int PGTYPESnumeric_from_double(double d, numeric* dst)	Converts a double-precision variable into a numeric variable.	This function accepts a double-precision variable and stores the result in the variable that **dst** points to. The function returns **0** on success and **–1** in case of error.
int PGTYPESnumeric_to_double(numeric* nv, double* dp)	Converts a numeric variable into a double-precision variable.	This function converts the numeric value in the variable that **nv** points to into a double-precision variable that **dp** points to. The function returns **0** on success and **–1** in case of error (or overflow). When overflow occurs, the global variable **errno** is additionally set to **PGTYPES_NUM_OVERFLOW**.

API	Description	Remarks
int PGTYPESnumeric_to_int(numeric* nv, int* ip)	Converts a numeric variable into an int variable.	This function converts the numeric value in the variable that **nv** points to into an int variable that the IP address points to. The function returns **0** on success and **–1** in case of error (or overflow). When overflow occurs, the global variable **errno** is additionally set to **PGTYPES_NUM_OVERFLOW**.
int PGTYPESnumeric_to_long(numeric* nv, long* ip)	Converts a numeric variable into a long int variable.	This function converts the numeric value in the variable that **nv** points to into a long int variable that the IP points to. The function returns **0** on success and **–1** in case of error (or overflow). When overflow occurs, the global variable **errno** is additionally set to **PGTYPES_NUM_OVERFLOW**.
int PGTYPESnumeric_to_decimal(numeric* src, decimal* dst)	Converts a numeric variable into a decimal variable.	This function converts the numeric value in the variable that **src** points to into a decimal variable that **dst** points to. The function returns **0** on success and **–1** in case of error (or overflow). When overflow occurs, the global variable **errno** is additionally set to **PGTYPES_NUM_OVERFLOW**.

API	Description	Remarks
int PGTYPESnumeric_from_decimal(decimal* src, numeric* dst)	Converts a decimal variable into a numeric variable.	This function converts the decimal value in the variable that **src** points to into a numeric variable that **dst** points to. The function returns **0** on success and **–1** in case of error (or overflow).

Examples

For details, see **Examples**.

5.8.16.3 Date Type

Table 5-94 lists the common date type APIs provided by ECPG.

Table 5-94 Common date type APIs

API	Description	Remarks
date* PGTYPESdate_new(void)	Returns a pointer to an allocated date variable.	This function creates a date variable on the heap. The return value is of the date* type.
void PGTYPESdate_free(date*)	Releases the memory that has been allocated to a date variable.	Frees a date* variable created using the PGTYPESdate_free function.

API	Description	Remarks
date PGTYPESdate_from_asc(char* str, char** endptr)	Parses a date from its textual representation.	This function accepts a C string **str** and a pointer to a C string **endptr**. ECPG converts the date expressed in text into a character string. Currently, ECPG does not support storing the address of the first invalid character in ***endptr**. However, **endptr** can be set to null. Note that the function always assumes MDY-formatted dates and there is currently no variable to change that within ECPG.
char* PGTYPESdate_to_asc(date dDate)	Returns the textual representation of a date variable.	This function accepts the date **dDate** as its unique parameter. It will output the date in the YYYY-MM-DD format.
date PGTYPESdate_from_timestamp(timestamp dt)	Extracts the date part from a timestamp.	This function accepts a timestamp as its unique parameter and returns the extracted date part from the timestamp.
void PGTYPESdate_julmdy(date jd, int* mdy)	Extracts the values of day, month, and year from a variable of the date type.	This function accepts the date **jd** and a pointer to an array of three integer values **mdy**. The variable name indicates the sequence. **mdy[0]** indicates the month, **mdy[1]** indicates the day, and **mdy[2]** indicates the year.

API	Description	Remarks
void PGTYPESdate_mdyjul(int * mdy, date* jdate)	Creates a date using the specified integer value.	This function accepts an array consisting of three integers (**mdy**) as its first parameter. The three integers are used to indicate the day, month, and year, respectively. The second parameter is a pointer to a variable of the date type, which is used to store the result of the operation.
int PGTYPESdate_dayofweek(date d)	Returns a number indicating the day of a week for a date value.	This function accepts the date variable **d** as its unique parameter and returns an integer indicating the day of the week.
void PGTYPESdate_today(date * d)	Returns the current date.	This function accepts a pointer to a date variable **d** and sets the parameter to the current date. • **0**: Sunday • **1**: Monday • **2**: Tuesday • **3**: Wednesday • **4**: Thursday • **5**: Friday • **6**: Saturday

API	Description	Remarks
int PGTYPESdate_defmt_asc(date* d, const char* fmt, char* str)	Converts a C char* string to a value of the date type using a format mask.	This function accepts a pointer **d** pointing to a date value for storing the operation result, a format mask **fmt** for parsing the date, and a C char* string **str** containing the textual representation of the date. The textual representation is expected to match the format mask. However, you do not need to have a 1:1 mapping of the string to the format mask. The function only analyzes the sequential order and looks for the literals "yy" or "yyyy" that indicate the position of the year, "mm" indicating the position of the month and "dd" indicating the position of the day. Example of valid input (fmt, str): yy/mm/dd 1994, February 3rd

API	Description	Remarks
int PGTYPESdate_fmt_asc(dat dDate, const char* fmtstring, char* outbuf)	Converts a variable of the date type into its textual representation using a format mask.	This function accepts the date **dDate** to be converted, the format mask **fmtstring**, and the string **outbuf** of the textual representation of the date to be saved. The function returns **0** on success and a negative value in case of error. The following is an example of valid input: 112359 //mmddyy 59/11/23 //yy/mm/dd Nov.23,1959 //mmm.dd,yyyy Mon,Nov.23,1959 // ddd,mmm.dd,yyyy Format: • **dd**: indicates the day of a month. • **mm**: indicates the month of a year. • **yy**: indicates a two-digit year. • **yyyy**: indicates a four-digit year. • **ddd**: indicates the day of a week. • **mmm**: indicates the month.

Examples

For details, see **Examples**.

5.8.16.4 Timestamp Type

Table 5-95 lists the common APIs for timestamp data provided by ECPG.

Table 5-95 Common timestamp type APIs

API	Description	Remarks
timestamp PGTYPEStimestamp_from_asc(char *str, char **endptr)	Parses a timestamp from its textual representation into a timestamp variable.	This function accepts a string **str** to parse and a pointer to a C char* **endptr**. It returns the parsed timestamp on success and **PGTYPESInvalidTimestamp** in case of error. In addition, **errno** is set to **PGTYPES_TS_BAD_TIMESTAMP** when an error occurs. The valid input of **PGTYPEStimestamp_from_asc** is as follows: 1999-01-08 04:05:06 January 8 04:05:06 1999 PST 1999-Jan-08 04:05:06.789-8 1999-01-08 04:05:06.789 (time zone specifier ignored) J2451187 04:05-08:00 1999-01-08 04:05:00 (time zone specifier ignored)
char *PGTYPEStimestamp_to_asc(timestamp tstamp)	Converts a date to a C char* string.	This function accepts the timestamp **tstamp** as its unique parameter and returns an allocated string containing the textual representation of the timestamp. The result must be released using PGTYPESchar_free().
void PGTYPEStimestamp_current(timestamp *ts)	Returns the current timestamp.	This function obtains the current timestamp and saves it to the timestamp variable that **ts** points to.

API	Description	Remarks
int PGTYPEStimestamp_fmt_asc(timestamp *ts, char *output, int str_len, char *fmtstr)	Converts a timestamp variable into a C char* using a format mask.	This function accepts a pointer pointing to the timestamp **ts** to be converted, a pointer pointing to the output buffer **output**, the maximum length allocated to the output buffer **str_len**, and the format mask **fmtstr** for conversion as its parameters. The function returns **0** on success and a negative value in case of error.
int PGTYPEStimestamp_sub(timestamp *ts1, timestamp *ts2, interval *iv)	Subtracts a timestamp from another timestamp and saves the result in an interval variable.	This function subtracts the timestamp variable that **ts2** points to from the timestamp variable that **ts1** points to, and stores the result in the interval variable that **iv** points to. The function returns **0** on success and a negative value in case of error.
int PGTYPEStimestamp_defmt_asc(char *str, char *fmt, timestamp *d)	Parses the timestamp value from its textual representation using a format mask.	This function accepts a timestamp textual representation placed in the variable **str** and a format mask placed in the variable **fmt** that will be used for conversion. The result will be stored in the variable that **d** points to. If the format mask **fmt** is null, the function rolls back to use the default format **mask %Y- %m-%d %H: %M: %S**.

API	Description	Remarks
int PGTYPEStimestamp_add_interval(timestamp *tin, interval *span, timestamp *tout)	Adds an interval variable to a timestamp variable.	This function accepts a pointer **tin** pointing to the timestamp variable and a pointer **span** pointing to the interval variable. It adds the interval to the timestamp and then saves the result timestamp in the variable that **tout** points to. The function returns **0** on success and a negative value in case of error.
int PGTYPEStimestamp_sub_interval(timestamp* tin, interval* span, timestamp* tout)	Subtracts an interval variable from a timestamp variable.	This function subtracts the interval variable that **span** points to from the timestamp variable that **tin** points to, and then saves the result in the variable that **tout** points to. The function returns **0** on success and a negative value in case of error.

Examples

For details, see **Examples**.

5.9 Compatibility Reference

JDBC Compatibility Package

Obtain the package from the release package **GaussDB-Kernel**_*Database version number*_*OS version number*_**64bit_Jdbc.tar.gz**.

After the decompression, you will obtain the following JDBC packages in .jar format:

- **gsjdbc4.jar**: The main class name is **org.postgresql.Driver**, and the URL prefix of the database connection is **jdbc:postgresql**. This driver package applies to the scenario where services are migrated from PostgreSQL. The driver class and loading path are the same as those before the migration, but the

supported APIs are different. The APIs that are not supported need to be adjusted on the service side.

- **gsjdbc200.jar**: This driver package applies to the scenario where services are migrated from GaussDB 200. The driver class and loading path are the same as those before the migration, but the supported interfaces are different. The interfaces that are not supported need to be adjusted on the service side.
- **opengaussjdbc.jar**: The main class name is **com.huawei.opengauss.jdbc.Driver**, and the URL prefix of the database connection is **jdbc:opengauss**. This driver package is used when both PostgreSQL and GaussDB are accessed in a JVM process.

NOTICE

- The loading paths and URL prefixes of driver classes vary in different driver packages, but the functions are the same.
- The **gsjdbc4** driver package cannot be used to operate the PostgreSQL database. Although the connection can be successfully established in some versions, some interface behaviors are different from those of PostgreSQL JDBC, which may cause unknown errors.
- The PostgreSQL driver package cannot be used to operate the GaussDB database. Although the connection can be successfully established in some versions, some interface behaviors are different from those of GaussDB JDBC, which may cause unknown errors.

Go Driver Compatibility

- The Go driver provided by the database does not adapt to mature ORM frameworks (such as XORM) in the industry. As such, the driver name input during database connection creation must be compatible with Postgres and PostgreSQL.
- After the environment variable *GOENV_REGIST_ALL* is set to **false**, the Go driver can coexist with that of PostgreSQL. If this parameter is not set or set to **true**, and the Go driver code depends on the Go driver of PostgreSQL, an error is reported.

ecpg Compatibility

ecpg provides the URL connection syntax which supports GaussDB and is compatible with PostgreSQL.

Connection syntax:

```
EXEC SQL CONNECT TO target [AS connection-name] [USER user-name];
```

To support PostgreSQL, **target** is set as follows:

- tcp:postgresql://*hostname*[:*port*][/*dbname*][?*options*]
- unix:postgresql://*hostname*[:*port*][/*dbname*][?*options*]

5.10 Commissioning

To control the output of log files and better understand the operating status of the database, modify specific configuration parameters in the **gaussdb.conf** file in the instance data directory.

Table 5-96 describes the adjustable configuration parameters.

Table 5-96 Configuration parameters

Parameter	Description	Value Range	Remarks
client_min_messages	Level of messages to be sent to clients.	• DEBUG5 • DEBUG4 • DEBUG3 • DEBUG2 • DEBUG1 • LOG • NOTICE • WARNING • ERROR • FATAL • PANIC Default value: **NOTICE**	Messages of the set level or lower will be sent to clients. The lower the level is, the fewer the messages will be sent.
log_min_messages	Level of messages to be recorded in server logs.	• DEBUG5 • DEBUG4 • DEBUG3 • DEBUG2 • DEBUG1 • INFO • NOTICE • WARNING • ERROR • LOG • FATAL • PANIC Default value: **WARNING**	Messages higher than the set level will be recorded in logs. The higher level indicates that the fewer server logs will be recorded.

Parameter	Description	Value Range	Remarks
log_min_error_statement	Level of SQL error statements to be recorded in server logs.	• DEBUG5 • DEBUG4 • DEBUG3 • DEBUG2 • DEBUG1 • INFO • NOTICE • WARNING • ERROR • FATAL • PANIC Default value: **ERROR**	SQL error statements of the set level or higher will be recorded in server logs. Only a system administrator is allowed to modify this parameter.
log_min_duration_statement	Minimum execution duration of a statement. If the execution duration of a statement is equal to or longer than the set milliseconds, the statement and its duration will be recorded in logs. Enabling this function can help you track the query attempts to be optimized.	INT type. Default value: **30min** Unit: millisecond	The value **–1** indicates that the function is disabled. Only a system administrator is allowed to modify this parameter.
log_connections/ log_disconnections	Specifies whether to record a server log message when each session is connected or disconnected.	• **on**: The system records a log server when each session is connected or disconnected. • **off**: The system does not record a log server when each session is connected or disconnected. Default value: **off**	-

Parameter	Description	Value Range	Remarks
log_duration	Specifies whether to record the duration of each executed statement.	• **on**: The system records the duration of each executed statement. • **off**: The system does not record the duration of each executed statement. Default value: **off**	Only a system administrator is allowed to modify this parameter.
log_statement	SQL statements to be recorded in logs.	• **none**: The system does not record any SQL statements. • **ddl**: The system records data definition statements. • **mod**: The system records data definition statements and data operation statements. • **all**: The system records all statements. Default value: **none**	Only a system administrator is allowed to modify this parameter.
log_hostname	Specifies whether to record host names.	• **on**: The system records host names. • **off**: The system does not record host names. Default value: **off**	By default, connection logs only record the IP addresses of connected hosts. With this function, the host names will also be recorded. This parameter has an impact on viewing audit results, **GS_SESSION_MEMORY_DETAIL**, **PG_STAT_ACTIVITY**, and the GUC parameter **log_line_prefix**.

Table 5-97 describes the preceding parameter levels.

Table 5-97 Description of log level parameters

Level	Description
DEBUG[1-5]	Provides information that can be used by developers. Level 1 is the lowest level whereas level 5 is the highest level.
INFO	Provides information about users' hidden requests, for example, information about the VACUUM VERBOSE process.
NOTICE	Provides information that may be important to users, for example, truncations of long identifiers or indexes created as a part of a primary key.
WARNING	Provides warning information for users, for example, COMMIT out of transaction blocks.
ERROR	Reports an error that causes a command to terminate.
LOG	Reports information that administrators may be interested in, for example, the activity levels of checkpoints.
FATAL	Reports the reason that causes a session to terminate.
PANIC	Reports the reason that causes all sessions to terminate.

6 SQL Optimization

The aim of SQL optimization is to maximize the utilization of resources, including CPU, memory, and disk I/O. All optimization methods are intended for resource utilization. To maximize resource utilization is to run SQL statements as efficiently as possible to achieve the highest performance at a lower cost. For example, when performing a typical point query, you can use a combination of Seq Scan and filter (that is, read each tuple and match the point query condition). You can also use Index Scan, which can implement the query at a lower cost but achieve the same effect.

You can determine a proper database deployment solution and table definition based on hardware resources and service characteristics. This is the basis of meeting performance requirements. The following performance tuning sections assume that you have finished installation based on a proper database solution in the software installation guide and performed database design based on the guide for database design and development.

6.1 Query Execution Process

The process from receiving SQL statements to the statement execution by the SQL engine is shown in **Figure 6-1** and described in **Table 6-1**. The texts in red are steps where database administrators can optimize queries.

Figure 6-1 Execution process of query-related SQL statements by the SQL engine

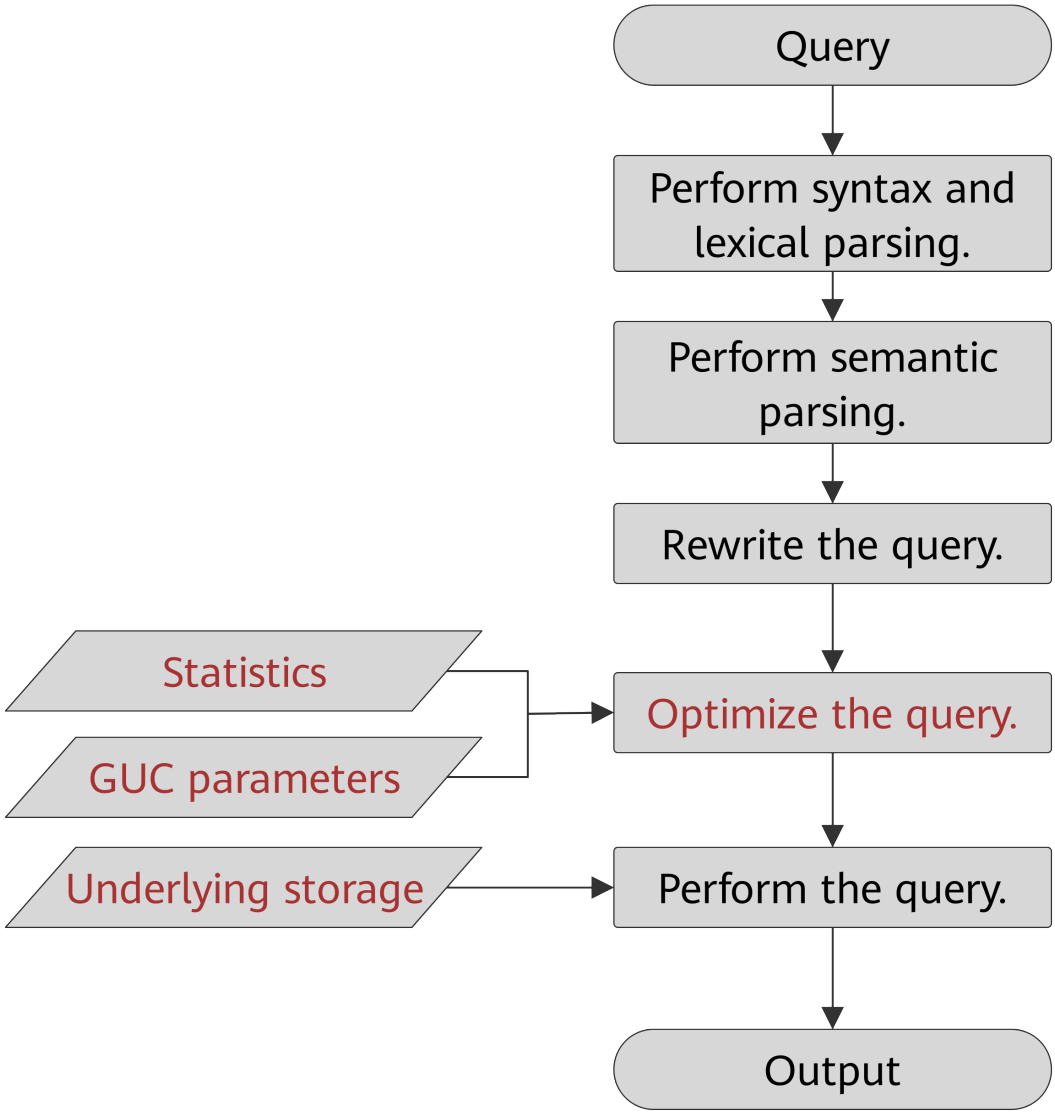


Table 6-1 Execution process of query-related SQL statements by the SQL engine

Step	Description
1. Perform syntax and lexical parsing.	Converts the input SQL statements from the string data type to the formatted structure stmt based on the specified SQL statement rules.
2. Perform semantic parsing.	Converts the formatted structure obtained from the previous step into objects that can be recognized by the database.
3. Rewrite the query statements.	Converts the output of the previous step into the structure that optimizes the query execution.

Step	Description
4. Optimize the query.	Determines the execution mode of SQL statements (the execution plan) based on the result obtained from the previous step and the internal database statistics. For details about how the internal database statistics and GUC parameters affect the query optimization (execution plan), see **Optimizing Queries Using Statistics** and **Optimizing Queries Using GUC parameters**.
5. Perform the query.	Executes the SQL statements based on the execution path specified in the previous step. Selecting a proper underlying storage mode improves the query execution efficiency.

Optimizing Queries Using Statistics

The GaussDB optimizer is a typical cost-based optimization (CBO). By using CBO, the database calculates the number of tuples and the execution cost for each step under each execution plan based on the number of table tuples, column width, null record ratio, and characteristic values, such as distinct, MCV, and HB values, and certain cost calculation methods. The database then selects the execution plan that takes the lowest cost for the overall execution or for the return of the first tuple. These characteristic values are the statistics, which is the core for optimizing a query. Accurate statistics helps the analyzer select the most appropriate query plan. Generally, you can collect statistics of a table or that of some columns in a table using ANALYZE. You are advised to periodically execute ANALYZE or execute it immediately after you modified most contents in a table.

Optimizing Queries Using GUC parameters

Optimizing queries aims to select an efficient execution mode.

Take the following SQL statement as an example:

```
select count(1)
from customer inner join store_sales on (ss_customer_sk = c_customer_sk);
```

During execution of **customer inner join store_sales**, GaussDB supports nested loops, merge joins, and hash joins. The optimizer estimates the result set size and the execution cost of each join mode based on the statistics on the **customer** and **store_sales** tables. It then compares the costs and selects the one costing the least.

As mentioned above, the execution cost is calculated based on certain methods and statistics. If the actual execution cost cannot be accurately estimated, you need to optimize the execution plan by setting GUC parameters. For example, the **random_page_cost** parameter indicates the optimizer's calculation of the cost of a non-sequentially-fetched disk page. The default value is **4**. When the random read speed of a machine disk (for example, an SSD) is high, you can decrease the value of this parameter. After the change, the cost of index scanning is reduced, and the index scanning mode is preferred when a plan is generated.

Optimizing Queries by Rewriting SQL Statements

Besides the preceding methods that improve the performance of the execution plan generated by the SQL engine, database administrators can also enhance SQL statement performance by rewriting SQL statements while retaining the original service logic based on the execution mechanism of the database and abundant practices.

This requires that database administrators know the customer services well and have professional knowledge of SQL statements. Below chapters will describe some common SQL rewriting scenarios.

6.2 Introduction to the SQL Execution Plan

6.2.1 Overview

The SQL execution plan is a node tree, which displays detailed procedure when GaussDB runs an SQL statement. A database operator indicates one step.

You can run the **EXPLAIN** command to view the execution plan generated for each query by an optimizer. The output of **EXPLAIN** has one row for each execution node, showing the basic node type and the cost estimation that the optimizer made for the execution of this node,

```
gaussdb=#  explain select * from t1,t2 where t1.c1 = t2.c2;
                           QUERY PLAN
-----------------------------------------------------------------------
 Hash Join  (cost=23.73..341.30 rows=16217 width=180)
   Hash Cond: (t1.c1 = t2.c2)
   ->  Seq Scan on t1  (cost=0.00..122.17 rows=5317 width=76)
   ->  Hash  (cost=16.10..16.10 rows=610 width=104)
         ->  Seq Scan on t2  (cost=0.00..16.10 rows=610 width=104)
(5 rows)
```

- Nodes at the bottom level are scan nodes. They scan tables and return raw rows. The types of scan nodes (sequential scans and index scans) vary depending on the table access methods. Objects scanned by the bottom layer nodes may not be row-store data (not directly read from a table), such as VALUES clauses and functions that return rows, which have their own types of scan nodes.
- If the query requires join, aggregation, sorting, or other operations on the raw rows, there will be other nodes above the scan nodes to perform these operations. In addition, there is more than one way to perform these operations, so different types of execution nodes may be displayed here.
- The first row (the upper-layer node) estimates the total execution cost of the execution plan. Such an estimate indicates the value that the optimizer tries to minimize.

Execution Plan Display Format

GaussDB provides four display formats: normal, pretty, summary, and run.

- normal: indicates that the default printing format is used.

- pretty: indicates that the new plan display format improved by GaussDB is used. The new format contains a plan node ID, directly and effectively analyzing performance.
- summary: indicates that the printing information analysis is added based on the pretty format.
- run: indicates that the information based on the summary format is exported as a CSV file for further analysis.

An example of an execution plan using the pretty format is as follows:

```
gaussdb=#  explain select * from t1,t2 where t1.c1=t2.c2;
 id |        operation         | E-rows | E-width |     E-costs
----+--------------------------+--------+---------+-----------------
  1 | ->  Hash Join (2,3)      |  23091 |      16 | 58.353..355.674
  2 |    ->  Seq Scan on t1    |   2149 |       8 | 0.000..31.490
  3 |    ->  Hash              |   2149 |       8 | 31.490..31.490
  4 |       ->  Seq Scan on t2 |   2149 |       8 | 0.000..31.490
(4 rows)

 Predicate Information (identified by plan id)
-----------------------------------------------
   1 --Hash Join (2,3)
         Hash Cond: (t1.c1 = t2.c2)
(2 rows)
```

You can change the display format of execution plans by setting the GUC parameter **explain_perf_mode**. The following uses the pretty format by default.

Execution Plan Information

In addition to setting different display formats for an execution plan, you can use different EXPLAIN syntax to display execution plan information in detail. The following lists the common EXPLAIN syntax. For details about more EXPLAIN syntax, see **EXPLAIN**.

- EXPLAIN *statement*: only generates an execution plan and does not execute. The *statement* indicates SQL statements.
- EXPLAIN ANALYZE *statement*: generates and executes an execution plan, and displays the execution summary. Then actual execution time statistics are added to the display, including the total elapsed time expended within each plan node (in milliseconds) and the total number of rows it actually returned.
- EXPLAIN PERFORMANCE *statement*: generates and executes the execution plan, and displays all execution information.

To measure the run time cost of each node in the execution plan, the current execution of EXPLAIN ANALYZE or EXPLAIN PERFORMANCE adds profiling overhead to query execution. Running EXPLAIN ANALYZE or EXPLAIN PERFORMANCE on a query sometimes takes longer time than executing the query normally. The amount of time that exceeds depends on the complexity of the query itself and the platform used.

Therefore, if an SQL statement is not finished after being running for a long time, run the **EXPLAIN** command to view the execution plan and then locate the fault. If the SQL statement has been properly executed, execute the EXPLAIN ANALYZE or EXPLAIN PERFORMANCE statement to check the execution plan and information to locate the fault.

6.2.2 Description

As described in **Overview**, EXPLAIN displays the execution plan, but will not actually run SQL statements. EXPLAIN ANALYZE and EXPLAIN PERFORMANCE both will actually run SQL statements and return the execution information. This section describes the execution plan and execution information in detail.

Execution Plans

The following SQL statement is used as an example:

```
SELECT * FROM t1, t2 WHERE t1.c1 = t2.c2;
```

Run the **EXPLAIN** command and the output is as follows:

```
gaussdb=#  EXPLAIN SELECT * FROM t1,t2 WHERE t1.c1 = t2.c2;
                          QUERY PLAN
------------------------------------------------------------------------
 Hash Join  (cost=23.73..341.30 rows=16217 width=180)
   Hash Cond: (t1.c1 = t2.c2)
   ->  Seq Scan on t1  (cost=0.00..122.17 rows=5317 width=76)
   ->  Hash  (cost=16.10..16.10 rows=610 width=104)
         ->  Seq Scan on t2  (cost=0.00..16.10 rows=610 width=104)
(5 rows)
```

Interpretation of the execution plan level (vertical):

1. Layer 1: **Seq Scan on t2**

   The table scan operator scans the table **t2** using **Seq Scan**. At this layer, data in the table **t2** is read from a buffer or disk, and then transferred to the upper-layer node for calculation.

2. Layer 2: **Hash**

   Hash operator. It is used to calculate the hash value of the operator transferred from the lower layer for subsequent hash join operations.

3. Layer 3: **Seq Scan on t1**

   The table scan operator scans the table **t1** using **Seq Scan**. At this layer, data in the table **t1** is read from a buffer or disk, and then transferred to the upper-layer node for hash join calculation.

4. Layer 4: **Hash Join**

   Join operator. It is used to join data in the **t1** and **t2** tables using the hash join method and output the result data.

Keywords in the execution plan:

1. Table access modes
   - Seq Scan

     Scans all rows of the table in sequence.
   - Index Scan

     The optimizer uses a two-step plan: the child plan node visits an index to find the locations of rows matching the index condition, and then the upper plan node actually fetches those rows from the table itself. Fetching rows separately is much more expensive than reading them sequentially, but because not all pages of the table have to be accessed, it is actually cheaper than a sequential scan. The upper-layer planning

node sorts index-identified rows based on their physical locations before reading them. This minimizes the independent capturing overhead.

If there are separate indexes on multiple columns referenced in **WHERE**, the optimizer might choose to use an **AND** or **OR** combination of the indexes. However, this requires the visiting of both indexes, so it is not necessarily a win compared to using just one index and treating the other condition as a filter.

The following index scans featured with different sorting mechanisms are involved:

- Bitmap index scan

  Fetches data pages using a bitmap.

- Index scan using index_name

  Uses simple index search, which fetches data from an index table in the sequence of index keys. This mode is commonly used when only a small amount of data needs to be fetched from a large data table or when the ORDER BY condition is used to match the index sequence to reduce the sorting time.

- Index-Only Scan

  Scans the index that contains all required data, instead of referencing a table.

– Bitmap Heap Scan

  Reads pages from bitmaps created by other operations and filters out the rows that do not meet the conditions. Bitmap heap scan can avoid random I/Os and accelerate read speed.

– TID Scan

  Scans a table by a tuple ID.

– Index Ctid Scan

  Scans a table based on the CTID index.

– CTE Scan

  Specifies that CTE evaluates subquery operations and stores query results as a temporary table. The temporary table is scanned by the CTE Scan operator.

– Foreign Scan

  Reads data from a remote data source.

– Function Scan

  Obtains result sets returned by functions and returns them as the rows read from tables.

– Sample Scan

  Queries and returns sampled data.

– Subquery Scan

  Reads subquery results.

– Values Scan

  Reads constants as part of the **VALUES** command.

- WorkTable Scan

  Scans a work table. Data is read in the middle of an operation which is usually a recursive operation declared using WITH RECURSIVE.

2. Table connection modes
   - Nested Loop

     A nested loop is used for queries that have a smaller dataset connected. In a nested loop join, the outer table drives the inner table and each row returned from the outer table should have a matching row in the inner table. The returned result set of all queries should be less than 10,000. The table that returns a smaller subset will work as an outer table, and indexes are recommended for connection columns of the inner table.

   - (Sonic) Hash Join

     A hash join is used for large tables. The optimizer uses a hash join, in which rows of one table are entered into an in-memory hash table, after which the other table is scanned and the hash table is probed for matches to each row. Sonic and non-Sonic hash joins differ in their hash table structures, which do not affect the execution result set.

   - Merge Join

     In most cases, the execution performance of a merge join is lower than that of a hash join. However, if the source data has been pre-sorted and no more sorting is needed during the merge join, its performance excels.

3. Operators
   - sort

     Sorts the result set.

   - filter

     The EXPLAIN output shows the WHERE clause being applied as a filter condition attached to the Seq Scan plan node. This means that the plan node checks the condition for each row it scans, and returns only the ones that meet the condition. The estimated number of output rows is reduced because of the WHERE clause. However, the scan will still have to visit all 10,000 rows, as a result, the cost is not decreased. It increases a bit (by 10,000 x **cpu_operator_cost**) to reflect the extra CPU time spent on checking the WHERE condition.

   - LIMIT

     Limits the number of output execution results. If a LIMIT condition is added, not all rows are retrieved.

   - Append

     Appends sub-operation results.

   - Aggregate

     Aggregates the results generated from querying rows. It can be an aggregation of statements such as GROUP BY, UNION, and SELECT DISTINCT.

   - BitmapAnd

     Specifies the AND operation of a bitmap, which is used to form a bitmap that matches more complex conditions.

   - BitmapOr

Specifies the OR operation of a bitmap, which is used to form a bitmap that matches more complex conditions.

- Gather

  Gathers data of parallel threads.

- Group

  Groups rows to perform the GROUP BY operation.

- GroupAggregate

  Aggregates the pre-sorted rows of the GROUP BY operation.

- Hash

  Hashes rows for the parent query. It is usually used to perform the JOIN operation.

- HashAggregate

  Aggregates the result rows of GROUP BY by using a hash table.

- Merge Append

  Merges subquery results in a way that preserves the sort order. It can be used to merge sorted rows in a table partition.

- ProjectSet

  Executes a function on the returned result set.

- Recursive Union

  Performs a union operation on all steps of a recursive function.

- SetOp

  Specifies a set operation, such as INTERSECT or EXCEPT.

- Unique

  Removes duplicates from an ordered result set.

- HashSetOp

  Specifies a strategy for set operations such as INTERSECT or EXCEPT. It uses Append to avoid pre-sorted input.

- LockRows

  Locks problematic rows to prevent other queries from writing, but allows reading.

- Materialize

  Stores subquery results in the memory so that the parent query can quickly access and obtain the subquery results.

- Result

  Returns a value without scanning.

- WindowAgg

  Specifies a window aggregate function, which is generally triggered by the OVER statement.

- Merge

  Performs a merge operation.

- StartWith Operator

  Specifies the hierarchical query operator, which is used to perform recursive query operations.

- Rownum

  Filters the row number in the query result. It usually appears in the ROWNUM clause.

- Index Cond

  Specifies the index scan conditions.

- Unpivot

  A transpose operator.

4. Partition pruning
   - Iterations

     Specifies the number of iterations performed by the partition iteration operator on level-1 partitions. If **PART** is displayed, dynamic pruning is used.

     For example, **Iterations: 4** indicates that the iteration operator needs to traverse four level-1 partitions. **Iterations: PART** indicates that the number of level-1 partitions to be traversed is determined by parameter conditions of the partition key.

   - Selected Partitions

     Specifies the selected level-1 partitions for pruning. **m..n** indicates that partitions **m** to **n** are selected. Multiple inconsecutive partitions are separated by commas (,).

     For example, **Selected Partitions: 2..4,7** indicates that partitions 2, 3, 4, and 7 are selected.

   - Sub Iterations

     Specifies the number of iterations performed by the partition iteration operator on level-2 partitions. If PART is displayed, dynamic pruning is used.

     For example, **Sub Iterations: 4** indicates that the iteration operator needs to traverse four level-2 partitions. **Iterations: PART** indicates that the number of level-2 partitions to be traversed is determined by parameter conditions of the partition key.

   - Selected Subpartitions

     Specifies the selected level-2 partitions for pruning, which is in the format like: level-1 partition number:level-2 partition number.

     For example, **Selected Subpartitions: 2:1 3:2** indicates that level-2 partition 1 of the second level-1 partition and level-2 partition 2 of the third level-1 partition are selected. **Selected Subpartitions: ALL** indicates that all level-2 partitions are selected.

5. Other keywords
   - Partitioned

     Indicates operations on a specific partition.

   - Partition Iterator

     Partition iterator, which usually indicates that a subquery is an operation on a partition.

   - InitPlan

     Indicates a non-related subplan.

Execution Information

The execution result of the following SQL statement in pretty mode is used as an example:

```
select sum(t2.c1) from t1,t2 where t1.c1=t2.c2 group by t1.c2;
```

The output of running EXPLAIN PERFORMANCE is as follows:

```
gaussdb=# explain performance select sum(t2.c1) from t1,t2 where t1.c1=t2.c2 group by t1.c2;
 id |            operation               | A-time | A-rows | E-rows | E-distinct | Peak Memory | A-width | E-width |
E-costs
----+------------------------------------+--------+--------+--------+------------+-------------+---------+---------
+-----------------
  1 | ->  HashAggregate                  | 0.574 |      0 |    200 |            | 29KB        |         |       8 |
396.113..398.113
  2 |    ->  Hash Join (3,4)             | 0.358 |      0 |  18915 | 200, 200   | 12KB        |         |       8 |
53.763..301.538
  3 |       ->  Seq Scan on public.t1    | 0.037 |      1 |   1945 |            | 22KB        |         |       8 | 0.000..29.450
  4 |       ->  Hash                     | 0.038 |      0 |   1945 |            | 264KB       |         |       8 | 29.450..29.450
  5 |         ->  Seq Scan on public.t2 | 0.029 |     30 |   1945 |            | 22KB        |         |       8 | 0.000..29.450
(5 rows)

 Predicate Information (identified by plan id)
-------------------------------------------------
  2 --Hash Join (3,4)
        Hash Cond: (t1.c1 = t2.c2)
(2 rows)

    Memory Information (identified by plan id)
-------------------------------------------------
  1 --HashAggregate
        Peak Memory: 29KB, Estimate Memory: 64MB
  2 --Hash Join (3,4)
        Peak Memory: 12KB, Estimate Memory: 64MB
  3 --Seq Scan on public.t1
        Peak Memory: 22KB, Estimate Memory: 64MB
  4 --Hash
        Peak Memory: 264KB
 Buckets: 32768  Batches: 1  Memory Usage: 0kB
  5 --Seq Scan on public.t2
        Peak Memory: 22KB, Estimate Memory: 64MB
(11 rows)

 Targetlist Information (identified by plan id)
-------------------------------------------------
  1 --HashAggregate
        Output: sum(t2.c1), t1.c2
        Group By Key: t1.c2
  2 --Hash Join (3,4)
        Output: t1.c2, t2.c1
  3 --Seq Scan on public.t1
        Output: t1.c1, t1.c2, t1.c3
  4 --Hash
        Output: t2.c1, t2.c2
  5 --Seq Scan on public.t2
        Output: t2.c1, t2.c2
(11 rows)

                          Datanode Information (identified by plan id)
-----------------------------------------------------------------------------------------------------------
  1 --HashAggregate
        (actual time=0.574..0.574 rows=0 loops=1)
        (Buffers: shared hit=2)
        (CPU: ex c/r=0, ex row=0, ex cyc=527797, inc cyc=8385141377087373)
  2 --Hash Join (3,4)
        (actual time=0.358..0.358 rows=0 loops=1)
        (Buffers: shared hit=2)
        (CPU: ex c/r=-8385141375712241, ex row=1, ex cyc=-8385141375712241, inc cyc=8385141376559576)
```



```
  3 --Seq Scan on public.t1
        (actual time=0.037..0.037 rows=1 loops=1)
        (Buffers: shared hit=1)
        (CPU: ex c/r=8385141375728512, ex row=1, ex cyc=8385141375728512, inc cyc=8385141375728512)
  4 --Hash
        (actual time=0.038..0.038 rows=0 loops=1)
        (Buffers: shared hit=1)
        (CPU: ex c/r=0, ex row=0, ex cyc=-251554241295571040, inc cyc=8385141376543305)
  5 --Seq Scan on public.t2
        (actual time=0.019..0.029 rows=30 loops=1)
        (Buffers: shared hit=1)
        (CPU: ex c/r=8664646089070478, ex row=30, ex cyc=259939382672114336, inc
cyc=259939382672114336)
(20 rows)

      ====== Query Summary =====
----------------------------------------
 Datanode executor start time: 0.180 ms
 Datanode executor run time: 0.590 ms
 Datanode executor end time: 0.051 ms
 Planner runtime: 0.366 ms
 Query Id: 844424930141239
 Total runtime: 0.866 ms
(6 rows)
```

In the preceding example, the execution information consists of the following parts:

1. The plan is displayed as a table, which contains 11 columns: **id**, **operation**, **A-time**, **A-rows**, **E-rows**, **E-distinct**, **Peak Memory**, **E-memory**, **A-width**, **E-width**, and **E-costs**. The meanings of the plan-type columns (**id**, **operation**, or columns started with **E**) are the same as those when EXPLAIN is executed. For details, see **Execution Plans**. The meanings of **A-time**, **A-rows**, **E-distinct**, **Peak Memory**, and **A-width** are described as follows:
   - **A-time**: execution completion time of the current operator.
   - **A-rows**: number of actual output tuples of the operator
   - **E-distinct**: estimated distinct value of the Hash Join operator.
   - **Peak Memory**: peak memory used by the operator during execution.
   - **A-width**: actual tuple width in each row of the current operator. This parameter is valid only for heavy memory operators, including (Vec)HashJoin, (Vec)HashAgg, (Vec)HashSetOp, (Vec)Sort, and (Vec)Materialize. The (Vec)HashJoin calculation width is the width of its right subtree operator and will be displayed on the right subtree.
2. **Predicate Information (identified by plan id)**:

   This part displays the static information that does not change in the plan execution process, such as some join conditions and filter information.
3. **Memory Information (identified by plan id)**:

   This part displays the memory usage information printed by certain operators (mainly Hash and Sort), including **peak memory**, **control memory**, **operator memory**, **width**, **auto spread num**, and **early spilled**; and spill details, including **spill Time(s)**, **inner/outer partition spill num**, **temp file num**, spilled data volume, and **written disk IO [***min, max***]**.
4. **Targetlist Information (identified by plan id)**:

   This part displays the target columns provided by each operator.
5. **DataNode Information (identified by plan id)**:

   This part displays the execution time, CPU, and buffer usage of each operator.

6. **====== Query Summary =====**:

   This part displays the total execution time and network traffic, including the maximum and minimum execution time in the initialization and end phases, available system memory when the current statement is executed, and estimated statement memory. The time-related statistics are described as follows:

   - **Datanode executor start time**: startup initialization time of the executor.
   - **Datanode executor run time**: running time of the executor.
   - **Datanode executor end time**: time when the executor stops clearing data.
   - **Planner runtime**: time when the optimizer generates a plan.
   - **Total runtime**: total execution time of the executor, including the time required for EXPLAIN. Therefore, **Total runtime** is greater than the sum of **Datanode executor start time**, **Datanode executor run time**, and **Datanode executor end time**.

   In addition, after the **\timing on** command is executed before the SQL statement is executed in the gsql command line, the total time from the statement to the display statement result is displayed at the end of the EXPLAIN result output.

6.3 Optimization Process

You can analyze slow SQL statements to optimize them.

Procedure

Step 1 Collect all table statistics associated with the SQL statements. In a database, statistics indicate the source data of a plan generated by an optimizer. If no collection statistics are available or out of date, the execution plan may seriously deteriorate, leading to low performance. According to past experience, about 10% performance problems occurred because no statistics are collected. For details, see **Updating Statistics**.

Step 2 View the execution plan to find out the cause. If the SQL statements have been running for a long period of time and not ended, run the **EXPLAIN** command to view the execution plan and then locate the fault. If the SQL statement has been properly executed, execute EXPLAIN ANALYZE or EXPLAIN PERFORMANCE to check the execution plan and information to locate the fault. For details about the execution plan, see **Introduction to the SQL Execution Plan**.

Step 3 Review and modify a table definition. For details, see **Reviewing and Modifying a Table Definition**.

Step 4 For details about EXPLAIN or EXPLAIN PERFORMANCE, the reason why SQL statements are slowly located, and how to solve this problem, see **Typical SQL Optimization Methods**.

Step 5 Generally, some SQL statements can be converted to its equivalent statements in all or certain scenarios by rewriting queries. SQL statements are simpler after they are rewritten. Some execution steps can be simplified to improve the performance. Query rewriting methods are universal in all databases. **Experience in Rewriting SQL Statements** describes several tuning methods by rewriting SQL statements.

Step 6 If the root cause of slow SQL statements cannot be analyzed using the preceding methods, you can use the plan trace feature to analyze the root cause of slow SQL statements. For details, see **Introduction to Plan Trace**.

----End

6.4 Updating Statistics

In a database, statistics indicate the source data of a plan generated by an optimizer. If no statistics are available or out of date, the execution plan may seriously deteriorate, leading to low performance.

Context

The ANALYZE statement collects statistic about table contents in databases, which will be stored in the PG_STATISTIC system catalog. Then, the query optimizer uses the statistics to work out the most efficient execution plan.

After executing batch insertions and deletions, you are advised to run the **ANALYZE** statement on the table or the entire library to update statistics. By default, 30,000 rows of statistics are sampled. That is, the default value of the GUC parameter **default_statistics_target** is **100**. If the total number of rows in the table exceeds 1,600,000, you are advised to set **default_statistics_target** to **-2**, indicating that 2% of the statistics are collected.

For an intermediate table generated during the execution of a batch script or stored procedure, you also need to run the ANALYZE statement.

If there are multiple inter-related columns in a table and the conditions or grouping operations based on these columns are involved in the query, collect statistics about these columns so that the query optimizer can accurately estimate the number of rows and generate an effective execution plan.

Procedure

Update the statistics about a table or the entire database.

```
ANALYZE tablename;                  -- Update statistics about a table.
ANALYZE;                            -- Update statistics about the entire database.
```

Perform statistics-related operations on multiple columns.

```
ANALYZE tablename ((column_1, column_2));                    -- Collect statistics about column_1 and column_2 of tablename.

ALTER TABLE tablename ADD STATISTICS ((column_1, column_2));    -- Declare statistics about column_1 and column_2 of tablename.
ANALYZE tablename;                                           -- Collect statistics about one or more columns.

ALTER TABLE tablename DELETE STATISTICS ((column_1, column_2)); -- Delete statistics about column_1 and column_2 of tablename or their statistics declaration.
```


NOTE

Use EXPLAIN to show the execution plan of each SQL statement. If **rows** is set to **10** (the default value, probably indicating that the table has not been analyzed) is displayed in the **SEQ SCAN** output of a table, execute the ANALYZE statement for this table.

After the statistics are declared for multiple columns by executing the ALTER TABLE tablename ADD STATISTICS statement, the system collects the statistics about these columns next time ANALYZE is performed on the table or the entire database.

To collect the statistics, run the **ANALYZE** command.

6.5 Reviewing and Modifying a Table Definition

6.5.1 Overview

To properly define a table, you must:

1. Reduce the data volume scanned by using the partition pruning mechanism.
2. Minimize random I/Os by using clustering.

The table definition is created during the database design and is reviewed and modified during the SQL statement optimization.

6.5.2 Using Partitioned Tables

Partitioning refers to splitting what is logically one large table into smaller physical pieces based on specific schemes. The table based on the logic is called a partitioned table, and a physical piece is called a partition. Data is stored in physical partitions not the logical table. A partitioned table has the following advantages over an ordinary table:

1. High query performance: You can specify partitions when querying partitioned tables, improving query efficiency.
2. High availability: If a certain partition in a partitioned table is faulty, data in the other partitions is still available.
3. Easy maintenance: To fix a partitioned table having a faulty partition, you only need to fix the partition.

Partitioned tables supported by the GaussDB database are level-1 and level-2 partitioned tables. Level-1 partitioned tables include range partitioned tables, interval partitioned tables, list partitioned tables, and hash partitioned tables. Level-2 partitioned tables include nine combinations of any two of range partitioned tables, list partitioned tables, and hash partitioned tables.

- Range partitioned table: Data in different ranges is mapped to different partitions. The range is determined by the partition key specified during the partitioned table creation. The partition key is usually a date. For example, sales data is partitioned by month.
- Interval partitioned table: a special type of range partitioned tables. Compared with range partitioned tables, interval value definition is added. When no matching partition can be found for an inserted record, a partition can be automatically created based on the interval value.
- List partitioned table: Key values contained in the data are stored in different partitions, and the data is mapped to each partition in sequence. The key

values contained in the partitions are specified when the partitioned table is created.

- Hash partitioned table: Data is mapped to each partition based on the internal hash algorithm. The number of partitions is specified when the partitioned table is created.
- Level-2 partitioned table: a partitioned table obtained by randomly combining range partitioning, list partitioning, and hash partitioning. Both level-1 and level-2 partitions can be defined in the preceding three ways.

6.5.3 Selecting a Data Type

Use the following principles to select efficient data types:

1. **Select data types that facilitate data calculation.**

   Generally, integer data calculations (including common comparison calculations such as =, >, <, ≥, ≤, and ≠, and GROUP BY) are more efficient than character strings and floating-point numbers.

2. **Select data types with a short length.**

   Data types with short length reduce both the data file size and the memory used for computing, improving the I/O and computing performance. For example, if smallint can be used for integer data, do not use int. If int can be used, do not use bigint.

3. **Use the same data type for a join.**

   You are advised to use the same data type for a join. To join columns with different data types, the database needs to convert them to the same type, which leads to additional performance overheads.

6.6 Typical SQL Optimization Methods

SQL optimization involves continuous analysis and trying. Queries are run before they are used for services to determine whether the performance meets requirements. If it does not, queries will be optimized by **checking the execution plan** and identifying the causes. Then, the queries will be run and optimized again until they meet the requirements.

6.6.1 Optimizing SQL Self-Diagnosis

Performance issues may occur when you query data or run the **INSERT**, **DELETE**, **UPDATE**, or **CREATE TABLE AS** statement. In this case, you can query the **warning** column in the **PG_CONTROL_GROUP_CONFIG** and **GS_SESSION_MEMORY_DETAIL** views to obtain reference for performance optimization.

Alarms that can trigger SQL self diagnosis depend on the settings of the GUC parameter **resource_track_level**. If **resource_track_level** is set to **query**, alarms about the failures in collecting column statistics and pushing down SQL statements will trigger the diagnosis. If **resource_track_level** is set to **operator**, all alarms will trigger the diagnosis.

Whether a SQL plan will be diagnosed depends on the settings of the GUC parameter **resource_track_cost**. An SQL plan will be diagnosed only if its

execution cost is greater than **resource_track_cost**. You can use the EXPLAIN keyword to check the plan execution cost.

The SQL self-diagnosis function is affected by the **enable_analyze_check** parameter. Ensure that the function is enabled before using it.

If a large number of statements are executed, certain data may fail to be collected due to memory control. In this case, you can increase the value of **instr_unique_sql_count**.

Alarms

Currently, performance alarms will be reported when statistics about one or multiple columns are not collected.

An alarm will be reported if some column statistics are not collected. For details about the optimization method, see **Updating Statistics** and **Optimizing Statistics**.

Example alarms:

No statistics about a table are not collected.

```
Statistic Not Collect:
    schema_test.t1
```

The statistics about a single column are not collected.

```
Statistic Not Collect:
    schema_test.t2(c1,c2)
```

The statistics about multiple columns are not collected.

```
Statistic Not Collect:
    schema_test.t3((c1,c2))
```

The statistics about a single column and multiple columns are not collected.

```
Statistic Not Collect:
    schema_test.t4(c1,c2)    schema_test.t4((c1,c2))
```

Restrictions

1. An alarm contains a maximum of 2048. If the length of an alarm exceeds this value (for example, a large number of long table names and column names are displayed in the alarm when their statistics are not collected), a warning instead of an alarm will be reported.
   ```
   WARNING, "Planner issue report is truncated, the rest of planner issues will be skipped"
   ```
2. If a query statement contains the **Limit** operator, alarms of operators lower than **Limit** will not be reported.

6.6.2 Optimizing Subqueries

Context

When an application runs an SQL statement to operate the database, a large number of subqueries are used because they are more clear than table join. Especially in complicated query statements, subqueries have more complete and independent semantics, which makes SQL statements clearer and easier to understand. Therefore, subqueries are widely used.

In GaussDB, subqueries can also be called sublinks based on the location of subqueries in SQL statements.

- Subquery: corresponds to a range table (RangeTblEntry) in the query parse tree. That is, a subquery is a SELECT statement following immediately after the FROM keyword.
- Sublink: corresponds to an expression in the query parsing tree. That is, a sublink is a statement in the **WHERE** or **ON** clause or in the target list.

  In conclusion, a subquery is a range table and a sublink is an expression in the query parsing tree. A sublink can be found in constraint conditions and expressions. In GaussDB, sublinks can be classified into the following types:

  - exist_sublink: corresponds to the EXIST and NOT EXIST statements.
  - any_sublink: corresponds to the *op* ANY(SELECT...) statement. *op* can be the <, >, or = operator. IN/NOT IN (SELECT...) also belongs to this type.
  - all_sublink: corresponds to the *op* ALL(SELECT...) statement. *op* can be the <, >, or = operator.
  - rowcompare_sublink: corresponds to the RECORD *op* (SELECT...) statement.
  - expr_sublink: corresponds to the (SELECT with a single target list item...) statement.
  - array_sublink: corresponds to the ARRAY(SELECT...) statement.
  - cte_sublink: corresponds to the WITH(...) query statement.

  The exist_sublink and any_sublink are pulled up by the optimization engine of GaussDB. In addition, expr_sublink can also be pulled up. However, because of the flexible use of subqueries in SQL statements, complex subqueries may affect query performance. If you do not want to pull up expr_sublink, set the GUC parameter **rewrite_rule**. Subqueries are classified into non-correlated subqueries and correlated subqueries.

  - **Non-correlated subqueries**

    The execution of a subquery is independent from attributes of the outer query. In this way, a subquery can be executed before outer queries.

    Example:

    ```
    gaussdb=# EXPLAIN SELECT t1.c1,t1.c2
    FROM t1
    WHERE t1.c1 IN (
       SELECT c2
       FROM t2
       WHERE t2.c2 IN (2,3,4)
    );
                                QUERY PLAN
    ------------------------------------------------------------------
     Hash Join
       Hash Cond: (t1.c1 = t2.c2)
       ->  Seq Scan on t1
             Filter: (c1 = ANY ('{2,3,4}'::integer[]))
       ->  Hash
             ->  HashAggregate
                   Group By Key: t2.c2
                   ->  Seq Scan on t2
                         Filter: (c2 = ANY ('{2,3,4}'::integer[]))
    (9 rows)
    ```

  - **Correlated subqueries**

    The execution of a subquery depends on some attributes (used as **AND** conditions of the subquery) of outer queries. In the following example,

t1.c1 in the **t2.c1 = t1.c1** condition is a correlated attribute. Such a subquery depends on outer queries and needs to be executed once for each outer query.

Example:

```
gaussdb=# EXPLAIN SELECT t1.c1,t1.c2
FROM t1
WHERE t1.c1 IN (
    SELECT c2
    FROM t2
    WHERE t2.c1 = t1.c1 AND t2.c2 IN (2,3,4)
);
                              QUERY PLAN
------------------------------------------------------------------------
 Seq Scan on t1
   Filter: (SubPlan 1)
   SubPlan 1
     ->  Seq Scan on t2
           Filter: ((c1 = t1.c1) AND (c2 = ANY ('{2,3,4}'::integer[])))
(5 rows)
```

Sublink Optimization on GaussDB

To optimize a sublink, a subquery is pulled up to join with tables in outer queries. You can run the EXPLAIN statement to check whether a sublink is converted into such a plan.

Example:

```
gaussdb=# EXPLAIN SELECT t1.c1, t1.c2 FROM t1 WHERE t1.c1 IN(SELECT c2 FROM t2 WHERE t2.c1 =
t1.c1);
          QUERY PLAN
---------------------------------
 Seq Scan on t1
   Filter: (SubPlan 1)
   SubPlan 1
     ->  Seq Scan on t2
           Filter: (c1 = t1.c1)
(5 rows)
```

- **Sublink-release scenarios supported by GaussDB**
  - Pulling up the **IN** sublink
    - The subquery cannot contain columns in the outer query (columns in more outer queries are allowed).
    - The subquery cannot contain volatile functions.

      Example:

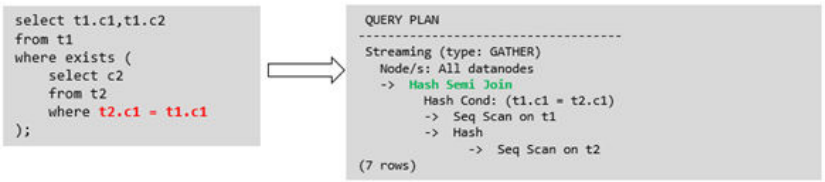
      ```
      gaussdb=# EXPLAIN SELECT t1.c1, t1.c2 FROM t1 WHERE t1.c1 IN(SELECT c2 FROM t2 WHERE
      t2.c1 = 1);
                 QUERY PLAN
      ---------------------------------------
       Hash Join
         Hash Cond: (t1.c1 = t2.c2)
         ->  Seq Scan on t1
         ->  Hash
               ->  HashAggregate
                     Group By Key: t2.c2
                     ->  Seq Scan on t2
                           Filter: (c1 = 1)
      (8 rows)
      ```

  - Pulling up the **EXISTS** sublink

The WHERE clause must contain a column in the outer query. Other parts of the subquery cannot contain the column. Other restrictions are as follows:

- The subquery must contain the FROM clause.
- The subquery cannot contain the **WITH** clause.
- The subquery cannot contain aggregate functions.
- The subquery cannot contain a **SET**, **SORT**, **LIMIT**, **WindowAgg**, or **HAVING** operation.
- The subquery cannot contain volatile functions.

  Example:



Replace the execution plan on the right of the arrow with the following execution plan:

```
          QUERY PLAN
----------------------------------
 Hash Join
   Hash Cond: (t1.c1 = t2.c1)
   ->  Seq Scan on t1
   ->  Hash
         ->  HashAggregate
               Group By Key: t2.c1
               ->  Seq Scan on t2
(7 rows)
```

– Pulling up an equivalent correlated query containing aggregate functions

  The WHERE condition of the subquery must contain a column from the outer query. Equivalence comparison must be performed between this column and related columns in tables of the subquery. These conditions must be connected using AND. Other parts of the subquery cannot contain the column. Other restrictions are as follows:

  - The columns in the expression in the WHERE condition of the subquery must exist in tables.
  - After the SELECT keyword of the subquery, there must be only one output column. The output column must be an aggregate function (for example, **MAX**), and the parameter (for example, **t2.c2**) of the aggregate function cannot be columns of a table (for example, **t1**) in outer queries. The aggregate function cannot be **COUNT**.

    For example, the following subquery can be pulled up:

    ```
    SELECT * FROM t1 WHERE c1 >(
          SELECT max(t2.c1) FROM t2 WHERE t2.c1=t1.c1
    );
    ```

    The following subquery cannot be pulled up because the subquery has no aggregate function:

```
SELECT * FROM t1 WHERE c1 >(
    SELECT  t2.c1 FROM t2 WHERE t2.c1=t1.c1
);
```

The following subquery cannot be pulled up because the subquery has two output columns:

```
SELECT * FROM t1 WHERE (c1,c2) >(
    SELECT max(t2.c1),min(t2.c2) FROM t2 WHERE t2.c1=t1.c1
);
```

- The subquery must be a FROM clause.
- The subquery cannot contain a GROUP BY, HAVING, or SET operation.
- The subquery can only be an INNER JOIN.

  For example, the following subquery cannot be pulled up:

  ```
  SELECT * FROM t1 WHERE c1 >(
      SELECT max(t2.c1) FROM t2 FULL JOIN t3 ON (t2.c2=t3.c2) WHERE t2.c1=t1.c1
  );
  ```

- The target list of the subquery cannot contain the function that returns a set.
- The **WHERE** condition of the subquery must contain a column from the outer query. Equivalence comparison must be performed between this column and related columns in tables of the subquery. These conditions must be connected using **AND**. Other parts of the subquery cannot contain the column. For example, the following subquery can be pulled up:

  ```
  SELECT * FROM t3 WHERE t3.c1=(
        SELECT t1.c1
        FROM t1 WHERE c1 >(
              SELECT max(t2.c1) FROM t2 WHERE t2.c1=t1.c1
  ));
  ```

  If another condition is added to the subquery in the previous example, the subquery cannot be pulled up because the subquery references to the column in the outer query. Example:

  ```
  SELECT * FROM t3 WHERE t3.c1=(
        SELECT t1.c1
        FROM t1 WHERE c1 >(
              SELECT max(t2.c1) FROM t2 WHERE t2.c1=t1.c1 AND t3.c1>t2.c2

  ));
  ```

– Pulling up a sublink in the **OR** clause

  If the WHERE condition contains an EXIST correlated sublink connected by OR, the following is an example:

  ```
  SELECT a, c FROM t1
  WHERE t1.a = (SELECT avg(a) FROM t3 WHERE t1.b = t3.b) OR
  EXISTS (SELECT * FROM t4 WHERE t1.c = t4.c);
  ```

  The process of pulling up such a sublink is as follows:

  i. Extract **opExpr** from the OR clause in the WHERE condition. The value is **t1.a = (select avg(a) from t3 where t1.b = t3.b)**.

  ii. The **opExpr** contains a subquery. If the subquery can be pulled up, the subquery is rewritten as **SELECT avg(a), t3.b FROM t3 GROUP BY t3.b**, generating the NOT NULL condition **t3.b IS NOT NULL**. The **opExpr** is replaced by this NOT NULL condition. In this case, the SQL statement changes to:


```
SELECT a, c
FROM t1 LEFT JOIN (SELECT avg(a) avg, t3.b FROM t3 GROUP BY t3.b) AS t3 ON (t1.a =
avg and t1.b = t3.b)
WHERE t3.b IS NOT NULL OR exists (SELECT * FROM t4 WHERE t1.c = t4.c);
```

iii. Extract the **EXISTS** sublink **exists (select * from t4 where t1.c = t4.c)** from the OR clause to check whether the sublink can be pulled up. If it can be pulled up, the subquery is converted into **select t4.c from t4 group by t4.c**, generating the NOT NULL condition **t4.c is not null**. In this case, the SQL statement changes to:

```
SELECT t1.a, t1.c FROM t1 LEFT JOIN (SELECT avg(a) avg, t3.b FROM t3 GROUP BY t3.b)
AS t3 ON (t1.a = avg AND t1.b = t3.b) LEFT JOIN (SELECT t5.c FROM t5 GROUP BY t5.c)
AS t5 ON (t1.c = t5.c) WHERE t3.b IS NOT NULL OR t5.c IS NOT NULL;
```

- **Sublink-release scenarios not supported by GaussDB**

  Except the sublinks described above, all the other sublinks cannot be pulled up. In this case, a join subquery is planned as the combination of subplans and broadcast. As a result, if inner tables have a large amount of data, query performance may be poor.

  If a correlated subquery joins with two tables in outer queries, the subquery cannot be pulled up. You need to change the parent query into a WITH clause and then perform the join.

  Example:

```
SELECT distinct t1.a, t2.a
FROM t1 LEFT JOIN t2 ON t1.a=t2.a AND NOT EXISTS (SELECT a,b FROM test1 WHERE test1.a=t1.a
AND test1.b=t2.a);
```

  The rewriting is as follows:

```
WITH temp AS
(
      SELECT * FROM (SELECT t1.a AS a, t2.a AS b FROM t1 LEFT JOIN t2 ON t1.a=t2.a)

)
SELECT distinct a,b
FROM temp
WHERE NOT EXISTS (SELECT a,b FROM test1 WHERE temp.a=test1.a AND temp.b=test1.b);
```

  – The subquery (without COUNT) in the target list cannot be pulled up.

    Example:

```
gaussdb=# EXPLAIN (costs off)
SELECT (SELECT c2 FROM t2 WHERE t1.c1 = t2.c1) ssq, t1.c2
FROM t1
WHERE t1.c2 > 10;
```

    The execution plan is as follows:

```
gaussdb=# EXPLAIN (costs off)
SELECT (SELECT c2 FROM t2 WHERE t1.c1 = t2.c1) ssq, t1.c2
FROM t1
WHERE t1.c2 > 10;
          QUERY PLAN
--------------------------------
 Seq Scan on t1
   Filter: (c2 > 10)
   SubPlan 1
     ->  Seq Scan on t2
           Filter: (t1.c1 = c1)
(5 rows)
```

    The correlated subquery is displayed in the target list (query return list). Values need to be returned even if the condition **t1.c1=t2.c1** is not met. Therefore, use a right outer join to join **t2** and **t1** so that the SSQ can return padding values when the condition **t1.c1=t2.c1** is not met.

> **NOTE**
>
> ScalarSubQuery (SSQ) and Correlated-ScalarSubQuery (CSSQ) are described as follows:
>
> - SSQ: a sublink that returns a scalar value of a single row with a single column
> - CSSQ: an SSQ containing correlation conditions

The preceding SQL statement can be changed into:

```
WITH ssq AS
(
   SELECT * FROM t1 WHERE t1.c2 >10
)
SELECT t2.c2,ssq.c2 FROM t2 RIGHT JOIN ssq ON ssq.c1 = t2.c1;
```

The execution plan after the change is as follows:

```
          QUERY PLAN
--------------------------------
 Hash Right Join
   Hash Cond: (t2.c1 = t1.c1)
   ->  Seq Scan on t2
   ->  Hash
         ->  Seq Scan on t1
               Filter: (c2 > 10)
(6 rows)
```

In the preceding example, the SSQ in the target list is pulled up to right join, preventing poor performance caused by the plan involving subplans when the table (**t2**) in the subquery is too large.

– The subquery (with COUNT) in the target list cannot be pulled up.

  Example:

```
SELECT (SELECT count(*) FROM t2 WHERE t2.c1=t1.c1) cnt, t1.c1, t3.c1
FROM t1,t3
WHERE t1.c1=t3.c1 ORDER BY cnt, t1.c1;
```

  The execution plan is as follows:

```
                QUERY PLAN
-------------------------------------------
 Sort
   Sort Key: ((SubPlan 1)), t1.c1
   ->  Hash Join
         Hash Cond: (t1.c1 = t3.c1)
         ->  Seq Scan on t1
         ->  Hash
               ->  Seq Scan on t3
         SubPlan 1
           ->  Aggregate
                 ->  Seq Scan on t2
                       Filter: (c1 = t1.c1)
(11 rows)
```

  The correlated subquery is displayed in the target list (query return list). Values need to be returned even if the condition **t1.c1=t2.c1** is not met. Therefore, use left outer join to join **T1** and **T2** so that SSQ can return padding values when the condition **t1.c1=t2.c1** is not met. However, COUNT is used, which requires that **0** is returned when the condition is not met. Therefore, case-when NULL then 0 else count(*) can be used.

  The preceding SQL statement can be changed into:

```
WITH ssq AS
(
   SELECT count(*) cnt, c1 FROM t2 GROUP BY c1
)
SELECT case WHEN
          ssq.cnt IS NULL THEN 0
```



```
        ELSE ssq.cnt
    END cnt, t1.c1, t3.c1
FROM t1 LEFT JOIN ssq ON ssq.c1 = t1.c1,t3
WHERE t1.c1 = t3.c1
ORDER BY ssq.cnt, t1.c1;
```

The execution plan after the change is as follows:

```
             QUERY PLAN
-----------------------------------------
 Sort
   Sort Key: ssq.cnt, t1.c1
   CTE ssq
     ->  HashAggregate
           Group By Key: t2.c1
           ->  Seq Scan on t2
   ->  Hash Join
         Hash Cond: (t1.c1 = t3.c1)
         ->  Hash Left Join
               Hash Cond: (t1.c1 = ssq.c1)
               ->  Seq Scan on t1
               ->  Hash
                     ->  CTE Scan on ssq
         ->  Hash
               ->  Seq Scan on t3
(15 rows)
```

– Non-equivalent correlated subqueries cannot be pulled up.

  Example:

```
SELECT t1.c1, t1.c2
FROM t1
WHERE t1.c1 = (SELECT agg() FROM t2.c2 > t1.c2);
```

  Non-equivalent correlated subqueries cannot be pulled up. You can perform join twice (one CorrelationKey and one rownum self-join) to rewrite the statement.

  You can rewrite the statement in either of the following ways:

  - Subquery rewriting:

```
SELECT t1.c1, t1.c2
FROM t1, (
    SELECT t1.rowid, agg() aggref
    FROM t1,t2
    WHERE t1.c2 > t2.c2 GROUP BY t1.rowid
) dt /* derived table */
WHERE t1.rowid = dt.rowid AND t1.c1 = dt.aggref;
```

  - CTE rewriting:

```
WITH dt as
(
    SELECT t1.rowid, agg() aggref
    FROM t1,t2
    WHERE t1.c2 > t2.c2 GROUP BY t1.rowid
)
SELECT t1.c1, t1.c2
FROM t1, dt
WHERE t1.rowid = dt.rowid AND
t1.c1 = dt.aggref;
```

NOTICE

- If the AGG type is COUNT(*), **0** is used for data padding when **CASE-WHEN** is not matched. If the type is not COUNT(*), **NULL** is used.
- CTE rewriting works better by using sharescan.

More Optimization Examples

Example: Modify the SELECT statement by modifying the subquery to a join relationship between the main table and the parent query or modifying the subquery to improve the query performance. Ensure that the subquery to be used is semantically correct.

```
gaussdb=# explain (costs off) select * from t1 where t1.c1 in (select t2.c1 from t2 where t1.c1 = t2.c2);
          QUERY PLAN
--------------------------------
 Seq Scan on t1
   Filter: (SubPlan 1)
   SubPlan 1
     ->  Seq Scan on t2
           Filter: (t1.c1 = c2)
(5 rows)
```

In the preceding example, a subplan is used. To remove the subplan, you can modify the statement as follows:

```
gaussdb=# explain (costs off) select * from t1 where exists (select t2.c1 from t2 where t1.c1 = t2.c2 and
t1.c1 = t2.c1);
               QUERY PLAN
-----------------------------------------
 Hash Join
   Hash Cond: (t1.c1 = t2.c2)
   ->  Seq Scan on t1
   ->  Hash
         ->  HashAggregate
               Group By Key: t2.c2, t2.c1
               ->  Seq Scan on t2
                     Filter: (c2 = c1)
(8 rows)
```

In this way, the subplan is replaced by the hash-join between the two tables, greatly improving the execution efficiency.

6.6.3 Optimizing Statistics

Context

GaussDB generates optimal execution plans based on the cost estimation. Optimizers need to estimate the number of data rows and the cost based on statistics collected using ANALYZE. Therefore, the statistics is vital for the estimation of the number of rows and cost. Global statistics are collected using ANALYZE: **relpages** and **reltuples** in pg_class; **stadistinct**, **stanullfrac**, **stanumbersN**, **stavaluesN**, and **histogram_bounds** in pg_statistic.

Example 1: Poor Query Performance Due to the Lack of Statistics

In most cases, the lack of statistics about tables or columns involved in the query greatly affects the query performance.

The table structure is as follows:

```
CREATE TABLE LINEITEM
(
L_ORDERKEY        BIGINT        NOT NULL
, L_PARTKEY       BIGINT        NOT NULL
, L_SUPPKEY       BIGINT        NOT NULL
, L_LINENUMBER    BIGINT        NOT NULL
, L_QUANTITY      DECIMAL(15,2) NOT NULL
```



```
, L_EXTENDEDPRICE  DECIMAL(15,2) NOT NULL
, L_DISCOUNT       DECIMAL(15,2) NOT NULL
, L_TAX            DECIMAL(15,2) NOT NULL
, L_RETURNFLAG     CHAR(1)       NOT NULL
, L_LINESTATUS     CHAR(1)       NOT NULL
, L_SHIPDATE       DATE          NOT NULL
, L_COMMITDATE     DATE          NOT NULL
, L_RECEIPTDATE    DATE          NOT NULL
, L_SHIPINSTRUCT   CHAR(25)      NOT NULL
, L_SHIPMODE       CHAR(10)      NOT NULL
, L_COMMENT        VARCHAR(44)   NOT NULL
);

CREATE TABLE ORDERS
(
O_ORDERKEY        BIGINT        NOT NULL
, O_CUSTKEY       BIGINT        NOT NULL
, O_ORDERSTATUS   CHAR(1)       NOT NULL
, O_TOTALPRICE    DECIMAL(15,2) NOT NULL
, O_ORDERDATE     DATE NOT NULL
, O_ORDERPRIORITY CHAR(15)      NOT NULL
, O_CLERK         CHAR(15)      NOT NULL
, O_SHIPPRIORITY  BIGINT        NOT NULL
, O_COMMENT       VARCHAR(79)   NOT NULL
);
```

The query statements are as follows:

```
explain verbose select
count(*) as numwait
from
lineitem l1,
orders
where
o_orderkey = l1.l_orderkey
and o_orderstatus = 'F'
and l1.l_receiptdate > l1.l_commitdate
and not exists (
select
*
from
lineitem l3
where
l3.l_orderkey = l1.l_orderkey
and l3.l_suppkey <> l1.l_suppkey
and l3.l_receiptdate > l3.l_commitdate
)
order by
numwait desc;
```

If such an issue occurs, you can use the following methods to check whether statistics in tables or columns has been collected using ANALYZE:

1. Execute EXPLAIN VERBOSE to analyze the execution plan and check the warning information:

   ```
   WARNING:Statistics in some tables or columns(public.lineitem.l_receiptdate,
   public.lineitem.l_commitdate, public.lineitem.l_orderkey, public.lineitem.l_suppkey,
   public.orders.o_orderstatus, public.orders.o_orderkey) are not collected.
   HINT:Do analyze for them in order to generate optimized plan.
   ```

2. Check whether the following information exists in the log file in the **gs_log** directory; if it does, the poor query performance was caused by the lack of statistics in some tables or columns:

   ```
   2017-06-14 17:28:30.336 CST 140644024579856 20971684 [BACKEND] LOG:Statistics in some tables
   or columns(public.lineitem.l_receiptdate, public.lineitem.l_commitdate, public.lineitem.l_orderkey,
   public.linei
   tem.l_suppkey, public.orders.o_orderstatus, public.orders.o_orderkey) are not collected.
   2017-06-14 17:28:30.336 CST 140644024579856 20971684 [BACKEND] HINT:Do analyze for them in
   order to generate optimized plan.
   ```

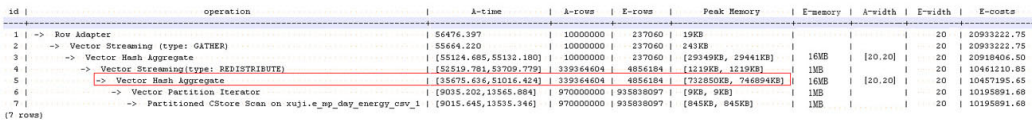

By using any of the preceding methods, you can identify tables or columns whose statistics have not been collected using ANALYZE. You can execute ANALYZE to warnings or tables and columns recorded in logs to resolve the problem.

6.6.4 Optimizing Operators

Context

A query statement needs to go through multiple operator procedures to generate the final result. Sometimes, the overall query performance deteriorates due to long execution time of certain operators, which are regarded as bottleneck operators. In this case, you need to run the **EXPLAIN ANALYZE** or **EXPLAIN PERFORMANCE** command to view the bottleneck operators, and then perform optimization.

For example, in the following execution process, the execution time of the Hashagg operator accounts for about 66% [(51016-13535)/56476 ≈ 66%] of the total execution time. Therefore, the Hashagg operator is the bottleneck operator for this query. Optimize this operator first.



id	operation	A-time	A-rows	E-rows	Peak Memory	E-memory	A-width	E-width	E-costs
1	-> Row Adapter	56476.397	10000000	237060	19KB			20	20933222.75
2	-> Vector Streaming (type: GATHER)	55664.220	10000000	237060	243KB			20	20933222.75
3	-> Vector Hash Aggregate	[55124.685,55132.180]	10000000	237060	[29349KB, 29441KB]	16MB	[20,20]	20	20918406.50
4	-> Vector Streaming(type: REDISTRIBUTE)	[52519.781,53709.779]	339364604	4856184	[1219KB, 1219KB]	1MB		20	10461210.85
5	-> Vector Hash Aggregate	[35675.636,51016.424]	339364604	4856184	[732850KB, 746894KB]	16MB	[20,20]	20	10457195.65
6	-> Vector Partition Iterator	[9035.202,13565.884]	970000000	935838097	[9KB, 9KB]	1MB		20	10195891.68
7	-> Partitioned CStore Scan on xuji.e_mp_day_energy_csv_1	[9015.645,13535.346]	970000000	935838097	[845KB, 845KB]	1MB		20	10195891.68

(7 rows)

Example

Example 1: Scan the base table. For queries requiring large volume of data filtering, such as point queries or queries that need range scanning, a full table scan using SeqScan will take a long time. To facilitate scanning, you can create indexes on the condition column and select IndexScan for index scanning.

```
gaussdb=#  explain (analyze on, costs off) select * from t1 where c1=10004;
 id |     operation      |  A-time  | A-rows | Peak Memory | A-width
----+--------------------+----------+--------+-------------+---------
  1 | ->  Seq Scan on t1 | 2053.069 |      7 | 64KB        |
(1 row)

 Predicate Information (identified by plan id)
-----------------------------------------------
   1 --Seq Scan on t1
         Filter: (c1 = 10004)
         Rows Removed by Filter: 110000
(3 rows)
gaussdb=#  create index idx on t1(c1);
CREATE INDEX
gaussdb=#  explain (analyze on, costs off) select * from t1 where c1=10004;
 id |           operation            | A-time | A-rows | Peak Memory | A-width
----+--------------------------------+--------+--------+-------------+---------
  1 | ->  Index Scan using idx on t1 | 0.227  |      7 | 77KB        |
(1 row)

 Predicate Information (identified by plan id)
-----------------------------------------------
   1 --Index Scan using idx on t1
         Index Cond: (c1 = 10004)
(2 rows)
```

In this example, the full table scan filters large amounts of data and returns 7 records. After an index has been created on the **c1** column, the scanning efficiency

is significantly boosted and the duration of IndexScan is reduced from 2s to 0.2 ms.

Example 2: If NestLoop is used for joining tables with a large number of rows, the join may take a long time. In the following example, NestLoop takes 27s. If **enable_mergejoin** is set to **off** to disable merge join and **enable_nestloop** is set to **off** to disable NestLoop so that the optimizer selects hash join, the join duration is reduced to 2s.

```
gaussdb=#  explain analyze select count(*) from t1,t2 where t1.c1=t2.c2;
 id |          operation          |  A-time   |  A-rows  | E-rows | Peak Memory | A-width | E-width |         E-
costs
----+-----------------------------+-----------+----------+--------+-------------+---------+---------
+-------------------------
  1 | ->  Aggregate               | 27544.545 |        1 |      1 | 15KB        |         |       8 | 1046289.218..1046289.228
  2 |    ->  Nested Loop (3,4)    | 27544.028 |     1992 |   3133 | 8KB         |         |       0 | 0.000..1046281.385
  3 |       ->  Seq Scan on t1    | 2043.884  |   110007 | 118967 | 61KB        |         |       4 | 0.000..157587.670
  4 |       ->  Materialize       | 6877.119  | 54783486 |    498 | 65KB        |         |       4 | 0.000..11.470
  5 |         ->  Seq Scan on t2 | 0.430     |      498 |    498 | 57KB        |         |       4 | 0.000..8.980
(5 rows)
```

After the parameters are set:

```
gaussdb=#  set enable_mergejoin=off;
SET
gaussdb=#  set enable_nestloop=off;
SET
gaussdb=#  explain analyze select count(*) from t1,t2 where t1.c1=t2.c2;
 id |          operation          |  A-time  | A-rows | E-rows | Peak Memory | A-width | E-width |        E-
costs
----+-----------------------------+----------+--------+--------+-------------+---------+---------+-------------------------
  1 | ->  Aggregate               | 2103.025 |      1 |      1 | 15KB        |         |       8 | 158088.164..158088.174
  2 |    ->  Hash Join (3,4)      | 2102.536 |   1992 |   3133 | 10KB        |         |       0 | 15.205..158080.331
  3 |       ->  Seq Scan on t1    | 2063.595 | 110007 | 118967 | 61KB        |         |       4 | 0.000..157587.670
  4 |       ->  Hash              | 0.753    |    498 |    498 | 296KB       |         |       4 | 8.980..8.980
  5 |         ->  Seq Scan on t2 | 0.480    |    498 |    498 | 57KB        |         |       4 | 0.000..8.980
(5 rows)
```

Example 3: Generally, query performance can be improved by selecting HashAgg. If Sort and GroupAgg are used for a large result set, you need to set **enable_sort** to **off**. HashAgg consumes less time than Sort and GroupAgg.

```
gaussdb=#  explain analyze select count(*) from t1 group by c1;
 id |        operation          |  A-time  | A-rows | E-rows | Peak Memory | A-width | E-width |        E-costs
----+---------------------------+----------+--------+--------+-------------+---------+---------+-------------------------
  1 | ->  GroupAggregate        | 2417.004 |  60000 |  18909 | 17KB        |         |      12 |
167616.708..168698.051
  2 |    ->  Sort               | 2304.329 | 200016 | 118967 | 26466KB     |         |       4 | 167616.708..167914.126
  3 |       ->  Seq Scan on t1 | 2125.464 | 200016 | 118967 | 58KB        |         |       4 | 0.000..157587.670
(3 rows)
```

After the parameters are set:

```
gaussdb=#  set enable_sort=off;
SET
gaussdb=#  explain analyze select count(*) from t1 group by c1;
 id |       operation        |  A-time  | A-rows | E-rows | Peak Memory | A-width | E-width |        E-costs
----+------------------------+----------+--------+--------+-------------+---------+---------+-------------------------
  1 | ->  HashAggregate      | 2324.062 |  60000 |  39912 | 10066KB     |         |       4 | 159297.545..159696.665
  2 |    ->  Seq Scan on t1 | 2131.319 | 200016 | 193303 | 58KB        |         |       4 | 0.000..158331.030
(2 rows)
```

6.7 Experience in Rewriting SQL Statements

Based on the SQL execution mechanism and a large number of practices, SQL statements can be optimized by following certain rules to enable the database to

execute SQL statements more quickly and obtain correct results. You can comply with these rules to improve service query efficiency.

- Use UNION ALL instead of UNION.

  UNION eliminates duplicate rows while merging two result sets but UNION ALL merges the two result sets without deduplication. Therefore, replace UNION with UNION ALL if you are sure that the two result sets do not contain duplicate rows based on the service logic.

- Add not null to the join columns.

  If there are many NULL values in the join columns, you can add the filter criterion IS NOT NULL to filter data in advance to improve the join efficiency.

- Convert NOT IN to NOT EXISTS.

  Nested loop anti join must be used to implement NOT IN, and hash anti join is required for NOT EXISTS. If no NULL value exists in the join columns, NOT IN is equivalent to NOT EXISTS. Therefore, if you are sure that no NULL value exists, you can convert NOT IN to NOT EXISTS to generate hash join and to improve the query performance.

  As shown in the following statement, if the **t2.d2** column does not contain NULL values (it is set to **NOT NULL**), the query can be modified as follows:

  ```
  SELECT * FROM t1 WHERE  NOT EXISTS (SELECT * FROM t2 WHERE t1.c1=t2.d2);
  ```

  The generated execution plan is as follows:

  ```
  QUERY PLAN
  ------------------------------
  Hash Anti Join
  Hash Cond: (t1.c1 = t2.d2)
  ->  Seq Scan on t1
  ->  Hash
  ->  Seq Scan on t2
  (5 rows)
  ```

- Use HashAgg.

  If a plan involving groupAgg and SORT operations generated by the GROUP BY statement is poor in performance, you can set **work_mem** to a larger value to generate a HashAgg plan, which does not require sorting and improves the performance.

- Replace functions with **CASE** statements.

  The GaussDB performance greatly deteriorates if a large number of functions are called. In this case, you can change the pushdown functions to CASE statements.

- Do not use functions or expressions for indexes.

  Using functions or expressions for indexes stops indexing. Instead, it enables the full table scan.

- Do not use the !=, <, or > operator, NULL, OR, or implicit parameter conversion in WHERE clauses.

  If the values of >= and <= are the same in WHERE condition, change the condition to = because range equivalence class derivation is not supported currently.

  ```
  SELECT * FROM t1 WHERE c1 >= 1 AND c1 <= 1;
  -- Change to:
  SELECT * FROM t1 WHERE c1 = 1;
  ```

  For range queries, the optimizer has a larger error when calculating selectivity than equivalent queries. Therefore, change range queries to equivalent queries as much as possible.

- Split complex SQL statements.

  You can split an SQL statement into several ones and save the execution result to a temporary table if the SQL statement is too complex to be tuned using the solutions above, including but not limited to the following scenarios:

  – The same subquery is involved in multiple SQL statements of a job and the subquery contains large amounts of data.
  – Incorrect plan cost causes a small hash bucket of subquery. For example, the actual number of rows is 10 million, but only 1000 rows are in hash bucket.
  – Functions such as substr and to_number cause incorrect measures for subqueries containing large amounts of data.

6.8 Configuring Key Parameters for SQL Tuning

This section describes key configuration parameters of the primary database node, which affect GaussDB SQL tuning. Contact the administrator to configure parameters.

Table 6-2 Parameters of the primary database node

Parameter/ Reference Value	Description
enable_nestloop=on	Specifies how the optimizer uses Nest Loop Join. If this parameter is set to **on**, the optimizer preferentially uses Nest Loop Join. If it is set to **off**, the optimizer preferentially uses other methods, if any. **NOTE** If you only want to temporarily change the value of this parameter during the current database connection (that is, the current session), execute the following SQL statement: `SET enable_nestloop to off;` By default, this parameter is set to **on**. Change the value as required. Generally, among the three join modes (Nested Loop, Merge Join, and Hash Join), Nested Loop is applicable to scenarios with small data volume or indexes, and Hash Join is applicable to big data analysis scenarios.

Parameter/ Reference Value	Description
enable_bitmapscan=on	Specifies whether the optimizer uses bitmap scan. If the value is **on**, bitmap scan is used. If the value is **off**, it is not used. **NOTE** If you only want to temporarily change the value of this parameter during the current database connection (that is, the current session), execute the following SQL statement: `SET enable_bitmapscan to off;` The bitmap scan applies only in the query condition **where a > 1 and b > 1** and indexes are created on columns **a** and **b**. However, the performance of bitmapscan is sometimes inferior to that of indexscan. During tuning, if the query performance is poor and bitmapscan operators are in the execution plan, set this parameter to **off** and check whether the performance is improved.
enable_hashagg=on	Specifies whether the optimizer uses hash aggregate plans.
enable_hashjoin=on	Specifies whether the optimizer uses hash join plans.
enable_mergejoin=on	Specifies whether the optimizer uses merge join plans.
enable_indexscan=on	Specifies whether the optimizer uses index scan plans.
enable_indexonlyscan=on	Specifies whether the optimizer uses index-only scan plans.
enable_seqscan=on	Specifies whether the optimizer uses sequential scan plans. It is impossible to suppress sequential scans entirely, but setting this variable to **off** encourages the optimizer to choose other methods if available.
enable_sort=on	Specifies the optimizer sorting order. It is impossible to fully suppress explicit sorting, but setting this variable to **off** encourages the optimizer to choose other methods if available.
rewrite_rule	Specifies whether the optimizer enables the LAZYAGG, MAGICSET, PARTIALPUSH, DISABLEREP, UNIQUECHECK, INTARGETLIST, PREDPUSH, PREDPUSHFORCE, PREDPUSHNORMAL, DISABLE_PULLUP_EXPR_SUBLINK, ENABLE_SUBLINK_PULLUP_ENHANCED, DISABLE_PULLUP_NOT_IN_SUBLINK, DISABLE_ROWNUM_PUSHDOWN, or DISABLE_WINDOWAGG_PUSHDOWN rewriting rule.

Parameter/ Reference Value	Description
sql_beta_feature	Specifies whether the optimizer enables the SEL_SEMI_POISSON, SEL_EXPR_INSTR, PARAM_PATH_GEN, RAND_COST_OPT, PARAM_PATH_OPT, PAGE_EST_OPT, CANONICAL_PATHKEY, PREDPUSH_SAME_LEVEL, PARTITION_FDW_ON, DISABLE_BITMAP_COST_WITH_LOSSY_PAGES, ENABLE_UPSERT_EXECUTE_GPLAN, or DISABLE_FASTPATH_INSERT beta feature.
var_eq_const_selectivity	Specifies whether the optimizer uses histograms to calculate the selectivity of integer constants.
partition_page_estimation	Specifies whether to optimize page estimation of partitioned table pages based on the pruning result. Only the partitioned table pages and local index pages are included. The global index pages are not included. The estimation formula is as follows: Number of pages after estimation = Total number of pages in the partitioned table x (Number of partitions after pruning/Total number of partitions)
partition_iterator_elimination	Specifies whether to eliminate the partition iteration operators to improve execution efficiency when the partition pruning result of a partitioned table is a partition.

Parameter/ Reference Value	Description
enable_functional_dependency	Specifies whether functional dependency statistics are used. Setting this parameter to **on** will enable the following functions: 1. The statistics about multiple columns generated by ANALYZE contain functional dependency statistics. 2. Function dependency statistics are used to calculate the selectivity. Setting this parameter to **off** will disable the following functions: 1. The statistics about multiple columns generated by ANALYZE do not contain functional dependency statistics. 2. Function dependency statistics are not used to calculate the selectivity. **NOTE** The concept of functional dependency comes from the relational database normal form, which indicates the functional relationship between attributes. The concept of functional dependency statistics derives from the preceding concept. It indicates the ratio of the data volume that meets the functional relationship to the total data volume. Functional dependency statistics are a type of multi-column statistics, which can be used to improve the accuracy of selectivity estimation. Functional dependency statistics are applicable to the **where a = 1 and b = 1** condition, in which **a** and **b** must be attributes of the same table subject to equivalent constraints and the constraints are connected by AND. There are at least two constraints.
enable_seqscan_fusion	Specifies whether to enable the sequential scan noise floor elimination function.
enable_inner_unique_opt	Specifies whether the optimizer uses Inner Unique.

6.9 Hint-based Tuning

6.9.1 Plan Hint Optimization

In plan hints, you can specify a join order, join and scan operations, and the number of rows in a result to tune an execution plan, improving query performance.

GaussDB also provides the SQL patch function. You can create an SQL patch to make hints take effect without modifying service statements.

Description

Plan hints are specified in the following format after keywords such as SELECT, INSERT, UPDATE, DELETE, and MERGE:

```
/*+ <plan hint>*/
```

You can specify multiple hints for a query plan and separate them with spaces. A hint specified for a query plan does not apply to its subquery plans. To specify a hint for a subquery, add the hint following the SELECT of this subquery.

Example:

```
select /*+ <plan_hint1> <plan_hint2> */ * from t1, (select /*+ <plan_hint3> */ * from t2) where 1=1;
```

In the preceding command, <*plan_hint1*> and <*plan_hint2*> are the hints of a query, and <*plan_hint3*> is the hint of its subquery.

You can use the EXPLAIN syntax to analyze the plan hint optimization effect. You can use EXPLAIN to view the plan of the target SQL statement after the plan hint is used and check whether the plan meets the requirements to verify the plan hint effect. EXPLAIN has multiple plan display modes, which are controlled by **explain_perf_mode**. In some examples in this section, **explain_perf_mode** is set to **pretty** to display complete plan information. In some examples, **explain_perf_mode** is set to **normal** to simplify the output information.

NOTICE

If a hint is specified in the CREATE VIEW statement, the hint will be applied each time this view is used.

If the random plan function is enabled (**plan_mode_seed** is set to a value other than 0), the specified hint will not be used.

Scope

Currently, the following hints are supported:

- Join order hints (**leading**).
- Join operation hints, excluding the **semi join**, **anti join**, and **unique plan** hints.
- Rows hints.
- Stream operation hints.
- Scan operation hints, supporting only the **tablescan**, **indexscan**, and **indexonlyscan** hints.
- Sublink name hints.
- Hints specifying not to expand subqueries.
- Hints for internal table materialization.
- Bitmap scan hints.
- Hints of the Agg method

Precautions

Hints do not support Sort, Setop, or Subplan.

Examples

The following are the statements (used in most examples in this section) and the original plan without hints for comparing the methods supported by Plan Hint:

```
create table t1(c1 int, c2 int, c3 int);
create table t2(c1 int, c2 int, c3 int);
create table t3(c1 int, c2 int, c3 int);
create index it1 on t1(c1,c2);
create index it2 on t2(c1,c2);
create index it3 on t1(c3,c2);
create table store
(
    s_store_sk              integer              not null,
    s_store_id              char(16)             not null,
    s_rec_start_date          date                       ,
    s_rec_end_date            date                       ,
    s_closed_date_sk          integer                    ,
    s_store_name              varchar(50)                ,
    s_number_employees        integer                    ,
    s_floor_space             integer                    ,
    s_hours                   char(20)                   ,
    s_manager                 varchar(40)                ,
    s_market_id               integer                    ,
    s_geography_class         varchar(100)               ,
    s_market_desc             varchar(100)               ,
    s_market_manager          varchar(40)                ,
    s_division_id             integer                    ,
    s_division_name           varchar(50)                ,
    s_company_id              integer                    ,
    s_company_name            varchar(50)                ,
    s_street_number           varchar(10)                ,
    s_street_name             varchar(60)                ,
    s_street_type             char(15)                   ,
    s_suite_number            char(10)                   ,
    s_city                    varchar(60)                ,
    s_county                  varchar(30)                ,
    s_state                   char(2)                    ,
    s_zip                     char(10)                   ,
    s_country                 varchar(20)                ,
    s_gmt_offset              decimal(5,2)               ,
    s_tax_precentage          decimal(5,2)               ,
    primary key (s_store_sk)
);
create table store_sales
(
    ss_sold_date_sk           integer                    ,
    ss_sold_time_sk           integer                    ,
    ss_item_sk                integer              not null,
    ss_customer_sk            integer                    ,
    ss_cdemo_sk               integer                    ,
    ss_hdemo_sk               integer                    ,
    ss_addr_sk                integer                    ,
    ss_store_sk               integer                    ,
    ss_promo_sk               integer                    ,
    ss_ticket_number          integer              not null,
    ss_quantity               integer                    ,
    ss_wholesale_cost         decimal(7,2)               ,
    ss_list_price             decimal(7,2)               ,
    ss_sales_price            decimal(7,2)               ,
    ss_ext_discount_amt       decimal(7,2)               ,
    ss_ext_sales_price        decimal(7,2)               ,
    ss_ext_wholesale_cost     decimal(7,2)               ,
    ss_ext_list_price         decimal(7,2)               ,
```

```
    ss_ext_tax               decimal(7,2)                ,
    ss_coupon_amt            decimal(7,2)                ,
    ss_net_paid              decimal(7,2)                ,
    ss_net_paid_inc_tax      decimal(7,2)                ,
    ss_net_profit            decimal(7,2)                ,
    primary key (ss_item_sk, ss_ticket_number)
);
create table store_returns
(
    sr_returned_date_sk      integer                     ,
    sr_return_time_sk        integer                     ,
    sr_item_sk               integer               not null,
    sr_customer_sk           integer                     ,
    sr_cdemo_sk              integer                     ,
    sr_hdemo_sk              integer                     ,
    sr_addr_sk               integer                     ,
    sr_store_sk              integer                     ,
    sr_reason_sk             integer                     ,
    sr_ticket_number         integer               not null,
    sr_return_quantity       integer                     ,
    sr_return_amt            decimal(7,2)                ,
    sr_return_tax            decimal(7,2)                ,
    sr_return_amt_inc_tax    decimal(7,2)                ,
    sr_fee                   decimal(7,2)                ,
    sr_return_ship_cost      decimal(7,2)                ,
    sr_refunded_cash         decimal(7,2)                ,
    sr_reversed_charge       decimal(7,2)                ,
    sr_store_credit          decimal(7,2)                ,
    sr_net_loss              decimal(7,2)                ,
    primary key (sr_item_sk, sr_ticket_number)
);
create table customer
(
    c_customer_sk            integer               not null,
    c_customer_id            char(16)              not null,
    c_current_cdemo_sk       integer                     ,
    c_current_hdemo_sk       integer                     ,
    c_current_addr_sk        integer                     ,
    c_first_shipto_date_sk   integer                     ,
    c_first_sales_date_sk    integer                     ,
    c_salutation             char(10)                    ,
    c_first_name             char(20)                    ,
    c_last_name              char(30)                    ,
    c_preferred_cust_flag    char(1)                     ,
    c_birth_day              integer                     ,
    c_birth_month            integer                     ,
    c_birth_year             integer                     ,
    c_birth_country          varchar(20)                 ,
    c_login                  char(13)                    ,
    c_email_address          char(50)                    ,
    c_last_review_date       char(10)                    ,
    primary key (c_customer_sk)
);
create table promotion
(
    p_promo_sk               integer               not null,
    p_promo_id               char(16)              not null,
    p_start_date_sk          integer                     ,
    p_end_date_sk            integer                     ,
    p_item_sk                integer                     ,
    p_cost                   decimal(15,2)               ,
    p_response_target        integer                     ,
    p_promo_name             char(50)                    ,
    p_channel_dmail          char(1)                     ,
    p_channel_email          char(1)                     ,
    p_channel_catalog        char(1)                     ,
    p_channel_tv             char(1)                     ,
    p_channel_radio          char(1)                     ,
    p_channel_press          char(1)                     ,
```



```
    p_channel_event           char(1)                   ,
    p_channel_demo            char(1)                   ,
    p_channel_details         varchar(100)              ,
    p_purpose                 char(15)                  ,
    p_discount_active         char(1)                   ,
    primary key (p_promo_sk)
);
create table customer_address
(
    ca_address_sk             integer               not null,
    ca_address_id             char(16)              not null,
    ca_street_number          char(10)                  ,
    ca_street_name            varchar(60)               ,
    ca_street_type            char(15)                  ,
    ca_suite_number           char(10)                  ,
    ca_city                   varchar(60)               ,
    ca_county                 varchar(30)               ,
    ca_state                  char(2)                   ,
    ca_zip                    char(10)                  ,
    ca_country                varchar(20)               ,
    ca_gmt_offset             decimal(5,2)              ,
    ca_location_type          char(20)                  ,
    primary key (ca_address_sk)
);
create table item
(
    i_item_sk                 integer               not null,
    i_item_id                 char(16)              not null,
    i_rec_start_date          date                      ,
    i_rec_end_date            date                      ,
    i_item_desc               varchar(200)              ,
    i_current_price           decimal(7,2)              ,
    i_wholesale_cost          decimal(7,2)              ,
    i_brand_id                integer                   ,
    i_brand                   char(50)                  ,
    i_class_id                integer                   ,
    i_class                   char(50)                  ,
    i_category_id             integer                   ,
    i_category                char(50)                  ,
    i_manufact_id             integer                   ,
    i_manufact                char(50)                  ,
    i_size                    char(20)                  ,
    i_formulation             char(20)                  ,
    i_color                   char(20)                  ,
    i_units                   char(10)                  ,
    i_container               char(10)                  ,
    i_manager_id              integer                   ,
    i_product_name            char(50)                  ,
    primary key (i_item_sk)
);
explain
select i_product_name product_name
,i_item_sk item_sk
,s_store_name store_name
,s_zip store_zip
,ad2.ca_street_number c_street_number
,ad2.ca_street_name c_street_name
,ad2.ca_city c_city
,ad2.ca_zip c_zip
,count(*) cnt
,sum(ss_wholesale_cost) s1
,sum(ss_list_price) s2
,sum(ss_coupon_amt) s3
FROM   store_sales
,store_returns
,store
,customer
,promotion
,customer_address ad2
```



```
,item
WHERE  ss_store_sk = s_store_sk AND
ss_customer_sk = c_customer_sk AND
ss_item_sk = i_item_sk and
ss_item_sk = sr_item_sk and
ss_ticket_number = sr_ticket_number and
c_current_addr_sk = ad2.ca_address_sk and
ss_promo_sk = p_promo_sk and
i_color in ('maroon','burnished','dim','steel','navajo','chocolate') and
i_current_price between 35 and 35 + 10 and
i_current_price between 35 + 1 and 35 + 15
group by i_product_name
,i_item_sk
,s_store_name
,s_zip
,ad2.ca_street_number
,ad2.ca_street_name
,ad2.ca_city
,ad2.ca_zip
;
 HashAggregate  (cost=18.09..18.10 rows=1 width=776)
   Group By Key: item.i_product_name, item.i_item_sk, store.s_store_name, store.s_zip, ad2.ca_street_number,
ad2.ca_street_name, ad2.ca_city, ad2.ca_zip
   ->  Nested Loop  (cost=0.00..18.06 rows=1 width=776)
         ->  Nested Loop  (cost=0.00..17.37 rows=1 width=416)
               ->  Nested Loop  (cost=0.00..17.08 rows=1 width=420)
                     ->  Nested Loop  (cost=0.00..16.67 rows=1 width=420)
                           ->  Nested Loop  (cost=0.00..16.26 rows=1 width=262)
                                 Join Filter: (item.i_item_sk = store_sales.ss_item_sk)
                                 ->  Nested Loop  (cost=0.00..15.46 rows=2 width=216)
                                       ->  Seq Scan on item  (cost=0.00..11.16 rows=1 width=208)
                                             Filter: ((i_current_price >= 35::numeric) AND (i_current_price <= 45::numeric)
AND (i_current_price >= 36::numeric) AND (i_current_price <= 50::numeric) AN
D (i_color = ANY ('{maroon,burnished,dim,steel,navajo,chocolate}'::bpchar[])))
                                       ->  Index Only Scan using store_returns_pkey on store_returns  (cost=0.00..4.29
rows=2 width=8)
                                             Index Cond: (sr_item_sk = item.i_item_sk)
                                 ->  Index Scan using store_sales_pkey on store_sales  (cost=0.00..0.38 rows=1
width=62)
                                       Index Cond: ((ss_item_sk = store_returns.sr_item_sk) AND (ss_ticket_number =
store_returns.sr_ticket_number))
                           ->  Index Scan using store_pkey on store  (cost=0.00..0.40 rows=1 width=166)
                                 Index Cond: (s_store_sk = store_sales.ss_store_sk)
                     ->  Index Scan using customer_pkey on customer  (cost=0.00..0.40 rows=1 width=8)
                           Index Cond: (c_customer_sk = store_sales.ss_customer_sk)
               ->  Index Only Scan using promotion_pkey on promotion  (cost=0.00..0.28 rows=1 width=4)
                     Index Cond: (p_promo_sk = store_sales.ss_promo_sk)
         ->  Index Scan using customer_address_pkey on customer_address ad2  (cost=0.00..0.68 rows=1
width=368)
               Index Cond: (ca_address_sk = customer.c_current_addr_sk)
(23 rows)
```

6.9.2 Hints for Specifying the Query Block Where the Hint Is Located

Description

This function allows users to use **@queryblock** in hints to implement block-level hint control. Users can specify the query block to which the hint takes effect. For example, you can specify the hint of an inner query block in the outer query block.

Syntax

Add **@queryblock** at the beginning of the hint parameter. **Hint_SEPC** is a specific hint.


```
Hint_SEPC([@queryblock])
```

Parameters

Hint_SEPC is the hint name, and **@queryblock** can be left empty. If **@queryblock** is left empty, the hint takes effect in the current query block declared by the hint. If **@queryblock** is left empty and **Hint_SPEC** has no parameter, use **Hint_SPEC** instead of **Hint_SPEC()**. Parentheses are unnecessary. The following describes how to name a query block and how to make a hint take effect. Some hints do not take effect only at the outermost layer and cannot be specified using **@queryblock**. For details, see the syntax description of each hint.

- Query the name of a query block.

  Each query block must have a name for accurately specifying a hint. There are two naming methods: user-specified and system-specified.

  a. You can use the blockname hint to specify the block to be queried. For details, see **Sublink Name Hints**.

  b. If no alias is specified for a query block, the default block name is automatically generated based on the processing sequence. Generally, the default alias of each query block consists of the first three letters of the query block name, **$**, and the number of the query block. For example, the alias of the first SELECT query block is **sel$1**. In pretty mode, you can use the explain method with a specified block name to view the name of the query block where the processing operator of each table is located.

  ```
  gaussdb=# set explain_perf_mode = pretty;
  SET
  gaussdb=# explain (blockname on,costs off) select * from t1, (select c1 from t2 group by c1)
  sub1 where t1.c1 = sub1.c1;
   id |             operation              | Query Block
  ----+------------------------------------+-------------
    1 | ->  Hash Join (2,3)                | sel$1
    2 |    ->  Seq Scan on t1@"sel$1"      | sel$1
    3 |    ->  Hash                        |
    4 |       ->  HashAggregate            | sel$2
    5 |          ->  Seq Scan on t2@"sel$2" | sel$2
  (5 rows)
  ```

  You can see that Seq Scan of **t2** is located in the **sel$2** query block.

- **@queryblock** specifies the query block.

  For the preceding example, if you want to modify the indexscan mode in **t2**, run the following command:

  ```
  select /*+indexscan(@sel$2 t2) tablescan(t1)*/ * from t1, (select c1 from t2 group by c1) sub1 where
  t1.c1 = sub1.c1;
  ```

  Both indexscan and tablescan are scan hints. For details about scan hints, see **Scan Hints**. You can specify the hint of indexscan(**@sel$2 t2**) in the **sel$1** query block to move the hint to the **sel$2** query block. The hint takes effect for **t2**. If the **sel$2** query block is promoted to **sel$1** during subsequent rewriting, the hint is also promoted together to **sel$1** and continues to take effect for **t2**.

  ```
  gaussdb=# explain (blockname on,costs off) select /*+indexscan(@sel$2 t2) tablescan(t1)*/ * from t1,
  (select c1 from t2 group by c1) sub1 where t1.c1 = sub1.c1;
   id |                   operation                    | Query Block
  ----+------------------------------------------------+-------------
    1 | ->  Hash Join (2,3)                            | sel$1
    2 |    ->  Seq Scan on t1@"sel$1"                  | sel$1
    3 |    ->  Hash                                    |
    4 |       ->  Group                                | sel$2
  ```

```
 5 |          ->  Index Only Scan using it2 on t2@"sel$2" | sel$2
(5 rows)
```

CAUTION

Sometimes, query rewriting in the optimizer phase expands some query blocks. As a result, the plan does not display related query blocks in the explain method. The hint specifies a query block based on the name of the query block before the optimizer phase. If a query block to be queried may be expanded in the planning phase, you can add the no_expand hint (see **Hints Specifying Not to Expand Subqueries**) to prevent it from being expanded.

1. The **sel$2** query block is a simple query. The optimizer performs query rewriting during subsequent processing, and **t1** is promoted to **sel$1** for processing. Therefore, the operation in the **sel$2** query block is not displayed in the plan.

```
gaussdb=# explain (blockname on,costs off) select * from t2, (select c1 from t1 where t1.c3 = 2)
sub1 where t2.c1 = sub1.c1;
 id |               operation                | Query Block
----+----------------------------------------+-------------
  1 | ->  Hash Join (2,3)                    | sel$1
  2 |    ->  Seq Scan on t2@"sel$1"          | sel$1
  3 |    ->  Hash                            |
  4 |       ->  Bitmap Heap Scan on t1@"sel$2" | sel$1
  5 |         ->  Bitmap Index Scan using it3 |
(5 rows)
```

2. The **sel$2** query block is a simple query. During subsequent processing, the optimizer skips query rewriting because of the no_expand hint, and **t1** is still processed in the original query block.

```
gaussdb=# explain (blockname on,costs off) select * from t2, (select /*+ no_expand*/ c1 from t1
where t1.c3 = 2) sub1 where t2.c1 = sub1.c1;
 id |               operation                | Query Block
----+----------------------------------------+-------------
  1 | ->  Hash Join (2,3)                    | sel$1
  2 |    ->  Seq Scan on t2@"sel$1"          | sel$1
  3 |    ->  Hash                            |
  4 |       ->  Bitmap Heap Scan on t1@"sel$2" | sel$2
  5 |         ->  Bitmap Index Scan using it3 |
(5 rows)
```

3. Because **t1** is processed in the **sel$2** query block after the no_expand hint is added, you can use **@sel$2** to specify the query block for the hint.

```
gaussdb=# explain (blockname on,costs off) select/*+ indexscan(@sel$2 t1)*/ * from t2, (select c1
from t1 where t1.c3 = 2) sub1 where t2.c1 = sub1.c1;
 id |                operation                | Query Block
----+-----------------------------------------+-------------
  1 | ->  Hash Join (2,3)                     | sel$1
  2 |    ->  Seq Scan on t2@"sel$1"           | sel$1
  3 |    ->  Hash                             |
  4 |       ->  Index Scan using it3 on t1@"sel$2" | sel$1
(4 rows)
```

4. The query block number in the view depends on the sequence of the statement using the view. Therefore, do not use hints to specify query blocks when creating a view. The behavior is uncontrollable.

```
gaussdb=# create view v1 as select/*+ no_expand */ c1 from t1 where c1 in (select /*+ no_expand
*/ c1 from t2 where t2.c3=4 );
CREATE VIEW
gaussdb=# explain (blockname on,costs off) select  * from v1;
 id |                operation                 | Query Block
----+------------------------------------------+-------------
  1 | ->  Seq Scan on t1@"sel$2"               | sel$2
  2 |    ->  Seq Scan on t2@"sel$3"  [1, SubPlan 1] | sel$3
(2 rows)

 Predicate Information (identified by plan id)
-----------------------------------------------
```



```
  1 --Seq Scan on t1@"sel$2"
        Filter: (hashed SubPlan 1)
  2 --Seq Scan on t2@"sel$3"
        Filter: (c3 = 4)
(4 rows)
```

In this case, the statements in **v1** belong to **sel$2** and **sel$3**.

5. Some hints do not take effect only at the outermost layer and cannot be specified using **@queryblock**. For details, see the syntax description of each hint.

6.9.3 Hints for Specifying the Query Block and Schema of a Table

Description

In a query, the table name can be duplicate in different query blocks and different schemas. Therefore, when specifying a table in a query, you can use a hint to specify the query block and schema where the table is located to avoid ambiguity. This function is applicable to all hints whose table names need to be specified.

Syntax

When specifying a table using a hint, specify a schema using a period (**schema.**) and a query block using the at sign (**@queryblock**). Both the schema and query block can be left blank.

```
[schema.]relname[@queryblock]
```

Parameters

- **relname** indicates the name of the table in the query. If the table has an alias, use the alias first. In this case, **relname** is set to the alias. If the table name contains special characters, such as at sign (@) and period (.), **relname** must be enclosed in double quotation marks ("") to avoid conflict with the declaration of the query block and schema names. For example, if the table name is **relnametest@1**, enter **"relnametest@1"**.
- **schema** indicates the schema where the table is located. It can be left empty. If no schema is specified, the hint searches all schemas for **relname**.
- **queryblock** indicates the query block where the table is located. It can be left empty. If no query block is specified, the hint searches all query blocks for **relname**.

Example

1. **t1** of **sel$2** is promoted to **sel$1** for processing, and **t1** is unclear.

```
gaussdb=# explain(blockname on,costs off) select /*+ indexscan(t1)*/ * from t1, (select c2 from t1
where c1=1) tt1 where t1.c1 = tt1.c2;
WARNING:  Error hint: IndexScan(t1), relation name "t1" is ambiguous.
...
```

2. **t1@sel$2** is specified to perform indexscan on **t1** of **sel$2** (Index Cond: (c1 = 1)).

```
gaussdb=# explain(blockname on,costs off) select /*+ indexscan(t1@sel$2)*/ * from t1, (select c2 from
t1 where c1=1) tt1 where t1.c1 = tt1.c2;
 id |                  operation                   | Query Block
```



```
----+-----------------------------------------------------+-------------
 1 | ->  Hash Join (2,3)                                 | sel$1
 2 |    ->  Seq Scan on t1@"sel$1"                       | sel$1
 3 |    ->  Hash                                         |
 4 |       ->  Index Only Scan using it1 on t1@"sel$2" | sel$1
(4 rows)

 Predicate Information (identified by plan id)
-------------------------------------------------
  1 --Hash Join (2,3)
        Hash Cond: (public.t1.c1 = public.t1.c2)
  4 --Index Only Scan using it1 on t1@"sel$2"
        Index Cond: (c1 = 1)
(4 rows)
```

6.9.4 Join Order Hints

Description

Specifies the join order and outer/inner tables.

Syntax

- Specify only the join order.

```
leading([@queryblock] join_table_list)
```

- Specify the join order and outer/inner tables. The outer/inner tables are specified by the outermost parentheses.

```
leading([@queryblock] (join_table_list))
```

Parameters

join_table_list specifies the tables to be joined. The values can be table names or table aliases. If a subquery is pulled up, the value can also be the subquery alias. Separate the values with spaces. You can add parentheses to specify the join priorities of tables.

For details about **@queryblock**, see **Hints for Specifying the Query Block Where the Hint Is Located**. **@queryblock** can be omitted, indicating that the hint takes effect in the current query block.

NOTICE

A table name or alias can only be a string without a schema name.

An alias (if any) is used to represent a table.

To prevent semantic errors, tables specified by **join_table_list** must meet the following requirements:

- The tables must exist in the query or its subquery to be pulled up.
- The table names must be unique in the query or subquery to be pulled up. If they are not, their aliases must be unique.
- A table appears only once in the list.
- An alias (if any) is used to represent a table.

For example:

leading(t1 t2 t3 t4 t5): **t1**, **t2**, **t3**, **t4**, and **t5** are joined. The join sequence and outer/inner tables are not specified.

leading((t1 t2 t3 t4 t5)): **t1**, **t2**, **t3**, **t4**, and **t5** are joined in sequence. The table on the right is used as the inner table in each join.

leading(t1 (t2 t3 t4) t5): First, **t2**, **t3**, and **t4** are joined and the outer/inner tables are not specified. Then, the result is joined with **t1** and **t5**, and the outer/ inner tables are not specified.

leading((t1 (t2 t3 t4) t5)): First, **t2**, **t3**, and **t4** are joined and the outer/inner tables are not specified. Then, the result is joined with **t1**, and **(t2 t3 t4)** is used as the inner table. Finally, the result is joined with **t5**, and **t5** is used as the inner table.

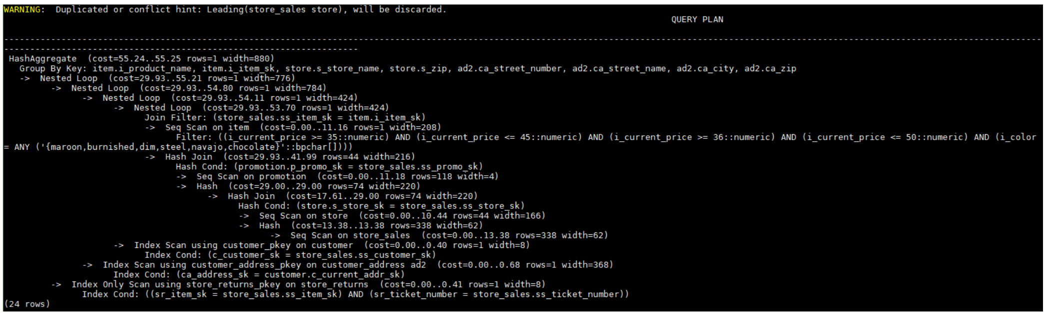
leading((t1 (t2 t3) t4 t5)) leading((t3 t2)): First, **t2** and **t3** are joined and **t2** is used as the inner table. Then, the result is joined with **t1**, and **(t2 t3)** is used as the inner table. Finally, the result is joined with **t4** and then **t5**, and the table on the right in each join is used as the inner table.

Examples

Hint the query plan in **Examples** as follows:

```
explain
select /*+ leading((((((store_sales store) promotion) item) customer) ad2) store_returns) leading((store
store_sales))*/ i_product_name product_name ...
```

First, **store_sales** and **store** are joined and **store_sales** is the inner table. Then, the result is joined with **promotion**, **item**, **customer**, **ad2**, and **store_returns** in sequence. The optimized plan is as follows.


```
WARNING:  Duplicated or conflict hint: Leading(store_sales store), will be discarded.
                                                                    QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=55.24..55.25 rows=1 width=880)
   Group By Key: item.i_product_name, item.i_item_sk, store.s_store_name, store.s_zip, ad2.ca_street_number, ad2.ca_street_name, ad2.ca_city, ad2.ca_zip
   ->  Nested Loop  (cost=29.93..55.21 rows=1 width=776)
         ->  Nested Loop  (cost=29.93..54.80 rows=1 width=784)
               ->  Nested Loop  (cost=29.93..54.11 rows=1 width=424)
                     ->  Nested Loop  (cost=29.93..53.70 rows=1 width=424)
                           Join Filter: (store_sales.ss_item_sk = item.i_item_sk)
                           ->  Seq Scan on item  (cost=0.00..11.16 rows=1 width=208)
                                 Filter: ((i_current_price >= 35::numeric) AND (i_current_price <= 45::numeric) AND (i_current_price >= 36::numeric) AND (i_current_price <= 50::numeric) AND (i_color
= ANY ('{maroon,burnished,dim,steel,navajo,chocolate}'::bpchar[])))
                           ->  Hash Join  (cost=29.93..41.99 rows=44 width=216)
                                 Hash Cond: (promotion.p_promo_sk = store_sales.ss_promo_sk)
                                 ->  Seq Scan on promotion  (cost=0.00..11.18 rows=118 width=4)
                                 ->  Hash  (cost=29.00..29.00 rows=74 width=220)
                                       ->  Hash Join  (cost=17.61..29.00 rows=74 width=220)
                                             Hash Cond: (store.s_store_sk = store_sales.ss_store_sk)
                                             ->  Seq Scan on store  (cost=0.00..10.44 rows=44 width=166)
                                             ->  Hash  (cost=13.38..13.38 rows=338 width=62)
                                                   ->  Seq Scan on store_sales  (cost=0.00..13.38 rows=338 width=62)
                     ->  Index Scan using customer_pkey on customer  (cost=0.00..0.40 rows=1 width=8)
                           Index Cond: (c_customer_sk = store_sales.ss_customer_sk)
               ->  Index Scan using customer_address_pkey on customer_address ad2  (cost=0.00..0.68 rows=1 width=368)
                     Index Cond: (ca_address_sk = customer.c_current_addr_sk)
         ->  Index Only Scan using store_returns_pkey on store_returns  (cost=0.00..0.41 rows=1 width=8)
               Index Cond: ((sr_item_sk = store_sales.ss_item_sk) AND (sr_ticket_number = store_sales.ss_ticket_number))
(24 rows)
```

For details about the warning at the top of the plan, see **Hint Errors, Conflicts, and Other Warnings**.

6.9.5 Join Operation Hints

Description

Specifies the join method, which can be nested loop join, hash join, or merge join.

Syntax

```
[no] nestloop|hashjoin|mergejoin([@queryblock] table_list)
```

Parameters

- For details about **@queryblock**, see **Hint for Specifying the Query Block Where the Hint Is Located**. This parameter can be omitted, indicating that it takes effect in the current query block.
- **no** indicates that the specified hint will not be used for a join.
- **table_list** specifies the tables to be joined. The values are the same as those of **join_table_list** but contain no parentheses.

For example:

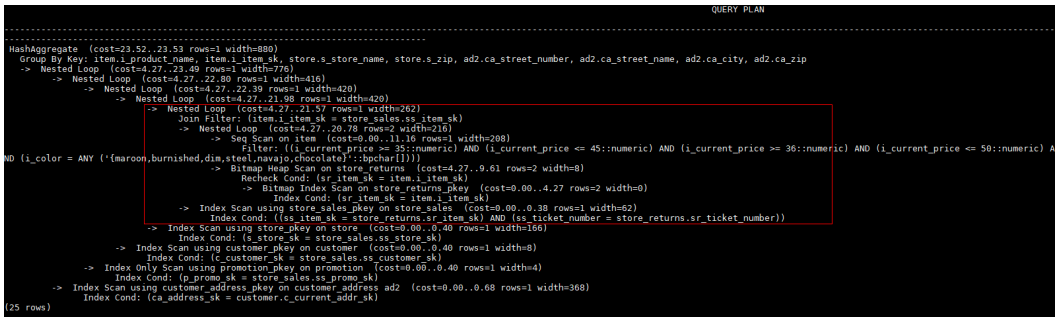
no nestloop(t1 t2 t3): **nestloop** is not used for joining **t1**, **t2**, and **t3**. The three tables may be joined in either of the two ways: Join **t2** and **t3**, and then **t1**; join **t1** and **t2**, and then **t3**. This hint takes effect only for the last join. If necessary, you can hint other joins. For example, you can add **no nestloop(t2 t3)** to join **t2** and **t3** first and to forbid the use of **nestloop**.

Example

Hint the query plan in **Examples** as follows:

```
explain
select /*+ nestloop(store_sales store_returns item) */ i_product_name product_name ...
```

nestloop is used for the last join between **store_sales**, **store_returns**, and **item**. The optimized plan is as follows.



6.9.6 Rows Hints

Description

Specifies the number of rows in an intermediate result set. Both absolute values and relative values are supported.

Syntax

```
rows( [@queryblock] table_list #|+|-|* const)
```

Parameters

- For details about **@queryblock**, see **Hint for Specifying the Query Block Where the Hint Is Located**. This parameter can be omitted, indicating that it takes effect in the current query block.
- #, +, -, and * are operators used for hinting the estimation. # indicates that the original estimation is used without any calculation. +, -, and * indicate

that the original estimation is calculated using these operators. The minimum calculation result is 1. *table_list* specifies the tables to be joined. The values are the same as those of **table_list** in **Join Operation Hints**.

- *const* can be any non-negative number and supports scientific notation.

For example:

rows(t1 #5): The result set of **t1** is five rows.

rows(t1 t2 t3 *1000): The result set of joined **t1**, **t2**, and **t3** is multiplied by 1000.

Suggestion

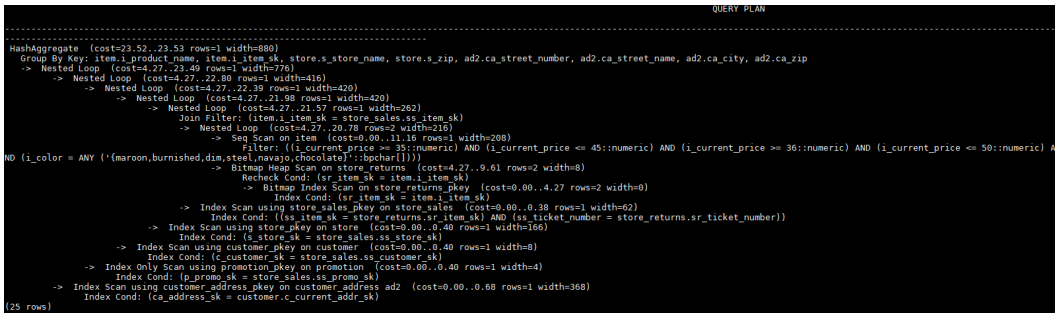
- The hint using * for two tables is recommended. This hint will be triggered if the two tables appear on two sides of a join. For example, if the hint is **rows(t1 t2 * 3)**, the join result of **(t1 t3 t4)** and **(t2 t5 t6)** will be multiplied by 3 because **t1** and **t2** appear on both sides of the join.
- **rows** hints can be specified for the result sets of a single table, multiple tables, function tables, and subquery scan tables.

Example

Hint the query plan in **Examples** as follows:

```
explain
select /*+ rows(store_sales store_returns *50) */ i_product_name product_name ...
```

Multiply the result set of joined **store_sales** and **store_returns** by 50. The optimized plan is as follows.



6.9.7 Stream Operation Hints

Description

A method of using stream in a parallel execution plan is specified. The value can be **broadcast** or **redistribute**, indicating that data is broadcast or redistributed.

Syntax

```
[no] broadcast|redistribute|local_roundrobin( [@queryblock] table_list)
```

Parameters

- For details about **@queryblock**, see **Hint for Specifying the Query Block Where the Hint Is Located**. This parameter can be omitted, indicating that it takes effect in the current query block.

- **broadcast**, **redistribute**, and **local_roundrobin** indicate data distribution methods.
  - **no** specifies that the specified hint will not be used for a stream operation.
  - **table_list** specifies one or more tables on which the stream operation is performed. Multiple tables are separated by spaces. For example, broadcast(t1) and broadcast(t1 t2).

Examples

```
create table stream_t1(a int, b int);
insert into stream_t1 values(generate_series(1, 1000000), generate_series(1, 1000000));
analyze stream_t1;
create table stream_t2(a int, b int);
insert into stream_t2 values(generate_series(1, 10000), generate_series(1, 10000));
analyze stream_t2;
set query_dop = 4;
explain (costs off) select/*+ broadcast(stream_t1)*/ * from stream_t1 join stream_t2 on (stream_t1.a =
stream_t2.a);
                        QUERY PLAN
------------------------------------------------------------------
 Streaming(type: LOCAL GATHER dop: 1/4)
   ->  Hash Join
         Hash Cond: (stream_t1.a = stream_t2.a)
         ->  Streaming(type: BROADCAST dop: 4/4)
               ->  Seq Scan on stream_t1
         ->  Hash
               ->  Streaming(type: LOCAL ROUNDROBIN dop: 4/1)
                     ->  Seq Scan on stream_t2
(8 rows)
-- Specify an execution plan to broadcast stream_t2.
explain (costs off) select/*+ broadcast(stream_t2)*/ * from stream_t1 join stream_t2 on (stream_t1.a =
stream_t2.a);
                     QUERY PLAN
-----------------------------------------------------------
 Streaming(type: LOCAL GATHER dop: 1/4)
   ->  Hash Join
         Hash Cond: (stream_t1.a = stream_t2.a)
         ->  Seq Scan on stream_t1
         ->  Hash
               ->  Streaming(type: BROADCAST dop: 4/1)
                     ->  Seq Scan on stream_t2
(7 rows)
-- The data of stream_t2 is broadcast and then joined with stream_t1. In this example, four concurrent
requests are enabled, and a table is broadcast to other threads for parallel hash join. The size of table
stream_t2 is smaller than that of table stream_t1. Therefore, broadcasting table t2 brings lower
performance overhead.

-- Specify an execution plan where stream_t2 uses local_roundrobin.
SET explain_perf_mode=pretty; -- Open the explain pretty option to view a detailed plan.
EXPLAIN (costs off) SELECT/*+ local_roundrobin(stream_t2)*/ * FROM stream_t1 JOIN stream_t2 ON
(stream_t1.a = stream_t2.a);
 id |                       operation
----+-------------------------------------------------------------
  1 | ->  Streaming(type: LOCAL GATHER dop: 1/4)
  2 |    ->  Hash Join (3,5)
  3 |       ->  Streaming(type: BROADCAST dop: 4/4)
  4 |          ->  Seq Scan on stream_t1
  5 |       ->  Hash
  6 |          ->  Streaming(type: LOCAL ROUNDROBIN dop: 4/1)
  7 |             ->  Seq Scan on stream_t2
(7 rows)

 Predicate Information (identified by plan id)
-----------------------------------------------
   2 --Hash Join (3,5)
```



```
      Hash Cond: (stream_t1.a = stream_t2.a)
(2 rows)
```

The **stream_t2** table uses the local_roundrobin data distribution method.

NOTE

The local_roundrobin hint takes effect only when the degree of parallelism for table scanning is 1. You are advised to use this hint with **scandop hint**.

Stream hints take effect only when parallel execution plans are generated.

6.9.8 Scan Operation Hints

Description

Specifies a scan operation, which can be **tablescan**, **indexscan**, or **indexonlyscan**.

Syntax

```
[no] tablescan|indexscan|indexonlyscan|gsi|gsitable( [@queryblock] table [index])
```

Parameters

- **no** indicates that the specified hint will not be used for a join.
- For details about **@queryblock**, see **Hint for Specifying the Query Block Where the Hint Is Located**. This parameter can be omitted, indicating that it takes effect in the current query block.
- **table** specifies the table to be scanned. You can specify only one table. Use a table alias (if any) instead of a table name.
- **index** indicates the index for **indexscan** or **indexonlyscan**. You can specify only one index.

NOTE

- **indexscan** and **indexonlyscan** hints can be used only when the specified index belongs to the table.
- Scan operation hints can be used for row-store tables, HDFS tables, and subquery tables.
- The index-only scan plan can be generated by the index scan hint, but the index-only hint can generate only the index-only plan.
- When index scan is compatible with index-only scan, some plan changes may occur. For example, **cost_model_version** is added for escape. This parameter can be used to determine whether index scan is compatible with index-only scan. Index scan is compatible with index-only scan when the parameter value is greater than 2 or equal to 0.
- The centralized environment does not support GSI and GSI table hints.

Example

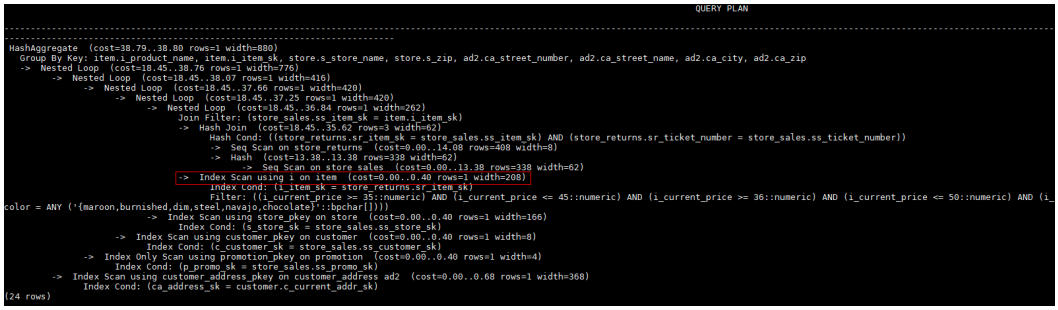
To specify an index-based hint for a scan, create an index named **i** on the **i_item_sk** column of the **item** table.

```
create index i on item(i_item_sk);
```

Hint the query plan in **Examples** as follows:

```
explain
select /*+ indexscan(item i) */ i_product_name product_name ...
```


item is scanned based on an index. The optimized plan is as follows.



In a centralized environment, when GSI hints are used, an alarm is reported indicating that GSI hints are not supported. The following is an example:

```
-- Create a table.
create table gsi_test(a int, b int);
create index gsi_test_idx on gsi_test(a);
-- Use indexes.
gaussdb=# explain select /*+  gsi(gsi_test gsi_test_idx) */ * from gsi_test where b = 1;
WARNING:  LINE 1: unsupport distributed hint at 'gsi'
                        QUERY PLAN
------------------------------------------------------------
 Seq Scan on gsi_test  (cost=0.00..36.86 rows=11 width=8)
   Filter: (b = 1)
(2 rows)
```

In a centralized environment, when GSI table hints are used, an alarm is reported indicating that GSI table hints are not supported. The following is an example:

```
gaussdb=# explain select /*+ gsitable(gsi_test gsi_test_idx) */ * from gsi_test where b = 1;
WARNING:  LINE 1: unsupport distributed hint at 'gsitable'
                        QUERY PLAN
------------------------------------------------------------
 Seq Scan on gsi_test  (cost=0.00..36.86 rows=11 width=8)
   Filter: (b = 1)
(2 rows)
```

6.9.9 Sublink Name Hints

Description

Specifies the name of a sublink block.

Syntax

```
blockname ( [@queryblock] table)
```

Parameters

- For details about **@queryblock**, see **Hint for Specifying the Query Block Where the Hint Is Located**. This parameter can be omitted, indicating that it takes effect in the current query block.
- **table** specifies the name you have specified for a sublink block.

NOTE

- The **blockname** hint is used by an outer query only when the corresponding sublink is not pulled up. Currently, only the **Agg** equivalent join, **IN**, and **EXISTS** sublinks can be pulled up. This hint is usually used together with the hints described in the previous sections.
- The subquery after the FROM keyword is hinted by using the subquery alias. In this case, the **blockname** hint becomes invalid.
- If a sublink contains multiple tables, the tables will be joined with the outer-query tables in a random sequence after the sublink is pulled up. In this case, the hint also becomes invalid.

Examples

```
explain select /*+nestloop(store_sales tt) */ * from store_sales where ss_item_sk in (select /*
+blockname(tt)*/ i_item_sk from item group by 1);
```

tt indicates the sublink block name. After being pulled up, the sublink is joined with the outer-query table **store_sales** by using **nestloop**. The optimized plan is as follows.

```
                                      QUERY PLAN
-------------------------------------------------------------------------------------------
 Nested Loop  (cost=10.53..68.39 rows=169 width=212)
   ->  HashAggregate  (cost=10.53..10.95 rows=42 width=4)
         Group By Key: item.i_item_sk
         ->  Seq Scan on item  (cost=0.00..10.42 rows=42 width=4)
   ->  Index Scan using store_sales_pkey on store_sales  (cost=0.00..1.34 rows=2 width=212)
         Index Cond: (ss_item_sk = item.i_item_sk)
(6 rows)
```

CAUTION

When the **blockname** hint is specified using **@queryblock** instead of taking effect in the current query block, for example, **blockname(@sel$2 new_qb_name)**, other hints cannot be specified using **@new_qb_name**. In this case, **new_qb_name** is only used as the name of the sublink and can be specified using hints.

- The block **bn2** is specified using **blockname(@sel$2 bn2)**. As a result, **TableScan(@bn2 t2)** cannot find the queryblock by using **@bn2**. The query block should be specified by using **@sel$2**. The block **bn3** is specified using the **blockname(bn3)** hint. The hint takes effect in the current query block and changes the name of the query block. Therefore, **tablescan(@bn3 t3@bn3)** can find the query block specified by using **@bn3**.

  ```
  gaussdb=# explain select /*+ blockname(@sel$2 bn2) tablescan(@bn2 t2) tablescan(@sel$2 t2@bn2)
  indexscan(@sel$2 t2@sel$2) tablescan(@bn3 t3@bn3)*/ c2 from t1 where c1 in ( select /*+ */t2.c1
  from t2 where t2.c2 = 1 group by 1) and c3 in ( select /*+ blockname(bn3)*/t3.c3 from t3 where t3.c2
  = 1 group by 1);
  WARNING:  hint: TableScan(@bn2 t2) does not match any query block
  WARNING:  Error hint: TableScan(@"sel$2" t2@bn2), relation name "t2@bn2" is not found.
  ```

- The following specifies the sublink block **bn2** by using **blockname(@sel$2 bn2)**. When the sublink is promoted, **hashjoin(t1 bn2)** can be used to specify the operation of the promoted sublink.

  ```
  gaussdb=# explain select /*+ blockname(@sel$2 bn2) hashjoin(t1 bn2) nestloop(t1 bn3) nestloop(t1
  sel$3)*/ c2 from t1 where c1 in ( select /*+ */t2.c1 from t2 where t2.c2 = 1 group by 1)  and c3 in
  ( select /*+ blockname(bn3)*/t3.c3 from t3 where t3.c2 = 1 group by 1);
  WARNING:  Duplicated or conflict hint: NestLoop(t1 "sel$3"), will be discarded.
  ```


6.9.10 Hint Errors, Conflicts, and Other Warnings

Plan hints change an execution plan. You can use EXPLAIN to view the changes.

Hints containing errors are invalid and do not affect statement execution. The errors will be displayed in different ways based on statement types. Hint errors in an EXPLAIN statement are displayed as a warning on the API. Hint errors in other statements will be recorded in debug1-level logs containing the PLANHINT keyword.

Hint error types are as follows:

- Syntax errors

  An error will be reported if the syntax tree fails to be reduced. The No. of the row generating an error is displayed in the error details.

  For example, the hint keyword is incorrect, no table or only one table is specified in the **leading** or **join** hint, or no tables are specified in other hints. The parsing of a hint is terminated immediately after a syntax error is detected. Only the hints that have been parsed successfully are valid.

  For example:

  ```
  leading((t1 t2)) nestloop(t1) rows(t1 t2 #10)
  ```

  The syntax of **nestloop(t1)** is wrong and its parsing is terminated. Only **leading(t1 t2)** that has been successfully parsed before **nestloop(t1)** is valid.

- Semantic errors
  - An error will be reported if the specified tables do not exist, multiple tables are found based on the hint setting, or a table is used more than once in the **leading** or **join** hint.
  - An error will be reported if the index specified in a scan hint does not exist.
  - If multiple tables with the same name exist after a subquery is pulled up and some of them need to be hinted, add aliases for them to avoid name duplication.

- Duplicated or conflicted hints

  If hint duplication or conflicts occur, only the first hint takes effect. A message will be displayed to describe the situation.
  - Hint duplication indicates that a hint is used more than once in the same query, for example, **nestloop(t1 t2) nestloop(t1 t2)**.
  - A hint conflict indicates that the functions of two hints with the same table list conflict with each other.

    For example, if **nestloop (t1 t2) hashjoin (t1 t2)** is used, **hashjoin (t1 t2)** becomes invalid. **nestloop(t1 t2)** does not conflict with **no mergejoin(t1 t2)**.

NOTICE

The table list in the **leading** hint is disassembled. For example, **leading ((t1 t2 t3))** will be disassembled as **leading((t1 t2)) leading(((t1 t2) t3))**, which will conflict with **leading((t2 t1))** (if any). If two hints use duplicated table lists and only one of them has the specified outer/inner table, the one without a specified outer/inner table becomes invalid.

Duplicate hints are allowed. However, for duplicate hints, only the first one is used. For other unused hints, an unused hint warning is reported. Take **/*+ expand_sublink expand_sublink */** as an example, database uses only the first expand_sublink hint. Therefore, the unused hint warning is still displayed.

- A hint becomes invalid after a sublink is pulled up.

  In this case, a message will be displayed. Generally, such invalidation occurs when a sublink contains multiple tables to be joined. After the sublink is pulled up, the tables will not be join members.
- Hints are not used.
  - If a **hashjoin** or **mergejoin** hint is specified for non-equivalent joins, it will not be used.
  - If an **indexscan** or **indexonlyscan** hint is specified for a table that does not have an index, it will not be used.
  - If an **indexscan** or **indexonlyscan** hint is specified for a full-table scan, it will not be used. Generally, index paths are generated only when filtering conditions are used on index columns. Indexes are not used during a full table scan.
  - If an **indexonlyscan** hint is specified when the output or predicate condition column does not contain only indexes, it will not be used.
  - In equivalent joins, only the joins containing equivalence conditions are valid. Therefore, the **leading**, **join**, and **rows** hints specified for the joins without an equivalence condition will not be used. For example, **t1**, **t2**, and **t3** are to be joined, and the join between **t1** and **t3** does not contain an equivalence condition. In this case, **leading(t1 t3)** will not be used.
  - If no sublink is pulled up, the specified **blockname** hint will not be used.

6.9.11 Optimizer GUC Parameter Hints

Description

Sets GUC parameters related to query optimization. The settings take effect during the query execution. For details about the application scenarios of hints, see the description of each GUC parameter.

Syntax

```
set( [@queryblock] param value)
```

Parameters

- For details about **@queryblock**, see **Hint for Specifying the Query Block Where the Hint Is Located**. It can be omitted, indicating that the hint takes

effect in the current query block. This hint takes effect only when it is specifies the outermost query block.

- **param** indicates the parameter name.
- **value** indicates the value of a parameter.
- Currently, the following parameters can be set and take effect by using hints:
  - Boolean

    enable_bitmapscan, enable_hashagg, enable_hashjoin, enable_indexscan, enable_indexonlyscan, enable_material, enable_mergejoin, enable_nestloop, enable_index_nestloop, enable_seqscan, enable_sort, enable_tidscan, partition_iterator_elimination, partition_page_estimation, enable_functional_dependency, enable_inner_unique_opt, enable_force_smp, enable_invisible_indexes, and var_eq_const_selectivity
  - Integer type

    query_dop and multi_insert_min_rows
  - Floating point

    cost_weight_index, default_limit_rows, seq_page_cost, random_page_cost, cpu_tuple_cost, cpu_index_tuple_cost, cpu_operator_cost, and effective_cache_size

NOTE

- If you set a parameter that is not in the whitelist and the parameter value is invalid or the hint syntax is incorrect, the query execution is not affected. Run **explain(verbose on)**. An error message is displayed, indicating that hint parsing fails.
- The GUC parameter hint takes effect only in the outermost query. That is, the GUC parameter hint in the subquery does not take effect.
- The GUC parameter hint in the view definition does not take effect.
- In the CREATE TABLE ... AS ... statement, the outermost GUC parameter hint takes effect.

6.9.12 Hints for Selecting the Custom Plan or Generic Plan

Description

For query statements and DML statements executed in PBE mode, the optimizer generates a custom plan or generic plan based on factors such as rules, costs, and parameters. You can use the hint of **use_cplan** or **use_gplan** to specify the plan to execute.

Syntax

- Specify that a custom plan is used.
  ```
  use_cplan
  ```
- Specify that a generic plan is used.
  ```
  use_gplan
  ```

NOTE

- For SQL statements that are executed in non-PBE mode, setting this hint does not affect the execution mode.
- This hint has a higher priority than cost-based selection and the **plan_cache_mode** parameter. That is, this hint does not take effect for statements where **plan_cache_mode** cannot forcibly select an execution mode.

Examples

Forcibly use the custom plan:

```
create table t (a int, b int, c int);
prepare p as select /*+ use_cplan */ * from t where a = $1;
explain execute p(1);
```

In the following plan, the filtering condition is the actual value of the input parameter, that is, the plan is a custom plan.

```
                        QUERY PLAN
-----------------------------------------------------
 Seq Scan on t  (cost=0.00..34.31 rows=10 width=12)
   Filter: (a = 1)
(2 rows)
```

Forcibly use the generic plan:

```
deallocate p;
prepare p as select /*+ use_gplan */ * from t where a = $1;
explain execute p(1);
```

In the following plan, the filtering condition is the input parameter to be added. That is, the plan is a generic plan.

```
                        QUERY PLAN
-----------------------------------------------------
 Seq Scan on t  (cost=0.00..34.31 rows=10 width=12)
   Filter: (a = $1)
(2 rows)
```

6.9.13 Hints Specifying Not to Expand Subqueries

Description

When the database optimizes the query logic, some subqueries can be pulled up to the upper layer to avoid nested execution. However, for some subqueries that have a low selectivity and can use indexes to filter pages, nested execution does not cause too much performance deterioration, while after the pull-up, the query search scope is expanded, which may cause performance deterioration. In this case, you can use the no_expand hint for debugging. This hint is not recommended in most cases.

Syntax

```
no_expand[(@queryblock)]
```

Parameters

For details about **@queryblock**, see **Hint for Specifying the Query Block Where the Hint Is Located**. This parameter can be omitted, indicating that it takes effect in the current query block. If it is not specified, **no_expand** does not have parentheses (()).

Example

Normal query execution:

```
explain select * from t1 where t1.c1 in (select t2.c1 from t2);
```

Plan

```
                                QUERY PLAN
---------------------------------------------------------------------------
 Hash Join  (cost=38.81..92.58 rows=972 width=12)
   Hash Cond: (t1.c1 = t2.c1)
   ->  Seq Scan on t1  (cost=0.00..29.45 rows=1945 width=12)
   ->  Hash  (cost=36.31..36.31 rows=200 width=4)
         ->  HashAggregate  (cost=34.31..36.31 rows=200 width=4)
               Group By Key: t2.c1
               ->  Seq Scan on t2  (cost=0.00..29.45 rows=1945 width=4)
(7 rows)
```

After **no_expand** is added:

```
explain select * from t1 where t1.c1 in (select /*+ no_expand*/ t2.c1 from t2);
```

Plan

```
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on t1  (cost=34.31..68.62 rows=972 width=12)
   Filter: (hashed SubPlan 1)
   SubPlan 1
     ->  Seq Scan on t2  (cost=0.00..29.45 rows=1945 width=4)
(4 rows)
```

6.9.14 Hint Specifying Not to Use Global Plan Cache

Function

When global plan cache is enabled, you can use the no_gpc hint to force a single query statement not to share the plan cache globally. Only the plan cache within the current session lifecycle is retained.

Syntax

```
no_gpc
```

NOTE

This parameter takes effect only for statements executed by PBE when **enable_global_plancache** is set to **on**.

Example

```
gaussdb=#  deallocate all;
DEALLOCATE ALL
gaussdb=#  prepare p1 as insert /*+ no_gpc*/ into t1 select c1,c2 from t2 where c1=$1;
PREPARE
gaussdb=#  execute p1(3);
INSERT 0 1
gaussdb=#  select * from dbe_perf.global_plancache_status where schema_name='public' order by 1,2;
```



```
 nodename | query | refcount | valid | databaseid | schema_name | params_num | func_id | pkg_id | stmt_id
----------+-------+----------+-------+------------+-------------+------------+---------+--------+---------
(0 rows)
```

No result exists in the **dbe_perf.global_plancache_status** view, that is, no plan is cached globally.

6.9.15 Hints of Parameterized Paths at the Same Level

Description

The predpush_same_level and nestloop_index hints are used to specify the generation of parameterized paths between tables or materialized views at the same level.

Syntax

```
predpush_same_level(src, dest)
predpush_same_level(src1 src2 ..., dest)
[no] nestloop_index([@queryblock] dest[, index_list]) -- With indexes
[no] nestloop_index([@queryblock] dest[,(src1 src2 ...)]) -- With tables
```

NOTE

The **predpush_same_level** parameter takes effect only when the **predpushforce** option in **rewrite_rule** is enabled.

nestloop_index has no requirement on **rewrite_rule**.

Parameters

- **no** indicates that the parameterized path of hints is not used.
- For details about **@queryblock**, see **Hint for Specifying the Query Block Where the Hint Is Located**. This parameter can be omitted, indicating that it takes effect in the current query block.
- **dest** is the target table of the parameterized path, that is, the table where the indexes are located.
- **src** is the parameter table of the parameterized path.
- **index_list** is the index sequence used by the parameterized path, which consists of character strings separated by spaces.

Examples

1. Examples of nestloop_index:

- Transfer **t2.c1** and **t3.c2** of **t2** and **t3** to the **t1** table for index scanning (parameterized path).

```
gaussdb=# explain (costs off) select /*+nestloop_index(t1,(t2 t3)) */* from t1,t2,t3 where t1.c1 = t2.c1 and t1.c2 = t3.c2;
                        QUERY PLAN
-----------------------------------------------------------
 Nested Loop
   ->  Seq Scan on t3
   ->  Nested Loop
         ->  Seq Scan on t2
         ->  Index Scan using it1 on t1
               Index Cond: ((c1 = t2.c1) AND (c2 = t3.c2))
(6 rows)
```


- Perform an index scan on **it1** of the **t1** table (parameterized path).

```
gaussdb=# explain (costs off) select /*+NestLoop_Index(t1,it1) */* from t1,t2 where t1.c1 = t2.c1;
           QUERY PLAN
---------------------------------
 Nested Loop
   ->  Seq Scan on t2
   ->  Index Scan using it1 on t1
         Index Cond: (c1 = t2.c1)
(4 rows)
```

2. Examples of predpush_same_level:

- Prepare parameters.

```
gaussdb=# set rewrite_rule = 'predpushforce';
SET
```

- View the plan.

```
gaussdb=# explain select * from t1, t2 where t1.c1 = t2.c1;
                          QUERY PLAN
------------------------------------------------------------------

 Hash Join  (cost=53.76..301.54 rows=18915 width=24)
   Hash Cond: (t1.c1 = t2.c1)
   ->  Seq Scan on t1  (cost=0.00..29.45 rows=1945 width=12)
   ->  Hash  (cost=29.45..29.45 rows=1945 width=12)
         ->  Seq Scan on t2  (cost=0.00..29.45 rows=1945 width=12)
(5 rows)
```

- The filter condition **t1.c1 = t2.c1** is displayed on **Join**. In this case, **predpush_same_level(t1, t2)** can be used to push the condition down to the scan operator of **t2**.

```
gaussdb=# explain select /*+predpush_same_level(t1, t2)*/ * from t1, t2 where t1.c1 = t2.c1;
                           QUERY PLAN
--------------------------------------------------------------------

 Nested Loop  (cost=0.00..1143.20 rows=18915 width=24)
   ->  Seq Scan on t1  (cost=0.00..29.45 rows=1945 width=12)
   ->  Index Scan using it2 on t2  (cost=0.00..0.47 rows=10 width=12)
         Index Cond: (c1 = t1.c1)
(4 rows)
```

NOTICE

- You can specify multiple **src** parameters in the same condition.
- If the specified **src** and **dest** conditions do not exist or do not meet the parameterized path requirements, this hint does not take effect.
- If a stream operator exists on the **dest** scanning operator, this hint does not take effect.

6.9.16 Hints for Setting Slow SQL Control Rules

Description

Users can set the execution time, maximum execution time, and maximum IOPS for SQL statements marked as slow SQL statements.

Syntax

```
wlmrule("time_limit,max_execute_time,max_iops")
```


NOTE

This parameter is valid only for SELECT statements executed by non-sysadmin or non-monitoradmin users when **enable_thread_pool** is set to **on**.

- **time_limit**: execution time of an SQL statement marked as a slow SQL statement. The value ranges from 0 to *INT_MAX*.
- **max_execute_time**: maximum execution time of an SQL statement. If the execution time exceeds the value of this parameter, the SQL statement is forcibly canceled and exits. The value ranges from **0** to *INT_MAX*. If the value of **max_execute_time** is less than or equal to the value of **time_limit**, the rule does not take effect.
- **max_iops**: maximum IOPS of an SQL statement marked as a slow SQL statement. This parameter is valid only when **use_workload_manager** is set to **on**. The IOPS limit applies logical I/O control. For details about the definition of IOPS, see the definition of **io_control_unit**. The value can be **Low**, **Medium**, **High**, **None**, or **0** to *INT_MAX*.

Examples

```
select /*+ wlmrule("100,500,1") */ * from t2 order by b limit 1;
```

It indicates that the execution time of the current statement marked as a slow SQL statement is 100 ms, the maximum execution time is 500 ms, and the maximum IOPS is 1.

6.9.17 Hint for Adaptive Plan Selection

Description

For query statements and DML statements executed in PBE mode, you can add the choose_adaptive_gplan hint to the query to trigger adaptive plan selection.

Syntax

Enable adaptive plan selection for query:

```
choose_adaptive_gplan
```

NOTE

- For SQL statements that are executed in non-PBE mode, setting this hint does not affect the execution mode.
- This hint takes effect only when the GUC parameter **enable_cachedplan_mgr** is enabled.

Example

```
prepare k as select /*+ choose_adaptive_gplan */ * from t1 where c1=$1 and c2=$2 and c3=$3 and c4=$4;
```

6.9.18 Hints for Materializing a Sub-plan Result

Description

You can materialize a sub-plan result to temporarily store the query record. This hint is used only in the INSERT statement.

When the INSERT INTO ... SELECT statement is used to insert a large amount of data with multiple duplicate rows, the index needs to be compared for multiple times. As a result, the execution takes a long time. This hint is used to materialize

the results of subplans and temporarily store query records to reduce the number of index comparisons and shorten the statement execution time.

Syntax

```
material_subplan
```

Examples

Create a table and insert data into the table.

```
create table test_mt_sub(a int, b int) with(storage_type = ustore);
create index on test_mt_sub(a);
create table test_src_mt_sub(a int, b int);
insert into test_src_mt_sub values(generate_series(1,10), generate_series(1,100000));
```

Normal INSERT INTO...SELECT statement:

```
insert into test_mt_sub select /*+ nestloop(test_src_mt_sub t1)*/ * from test_src_mt_sub where not
exists(select 1 from test_mt_sub t1 where t1.a = test_src_mt_sub.a);
```

The execution plan is as follows:

```
                        QUERY PLAN
------------------------------------------------------------------------
 Insert on test_mt_sub
   ->  Nested Loop Anti Join
         ->  Seq Scan on test_src_mt_sub
         ->  Index Only Scan using test_mt_sub_a_idx on test_mt_sub t1
               Index Cond: (a = test_src_mt_sub.a)
(5 rows)
```

Use the material_subplan hint operator.

```
insert /*+ material_subplan*/ into test_mt_sub select /*+ nestloop(test_src_mt_sub t1)*/ * from
test_src_mt_sub where not exists(select 1 from test_mt_sub t1 where t1.a = test_src_mt_sub.a);
```

The execution plan is as follows:

```
                              QUERY PLAN
-------------------------------------------------------------------------------
 Insert on test_mt_sub
   ->  Materialize
         ->  Nested Loop Anti Join
               ->  Seq Scan on test_src_mt_sub
               ->  Index Only Scan using test_mt_sub_a_idx on test_mt_sub t1
                     Index Cond: (a = test_src_mt_sub.a)
(6 rows)
```

6.9.19 Bitmap Scan Hints

Description

These hints generate a bitmap scan path by using the specified index on the target table. The path that meets the hint requirement is selected from the paths that can be generated by the optimizer.

Syntax

```
[no] bitmapscan([@queryblock] table [index_list])
```

Parameters

- **no** indicates that the specified hint will not be used for a join.

- For details about **@queryblock**, see **Hint for Specifying the Query Block Where the Hint Is Located**. This parameter can be omitted, indicating that it takes effect in the current query block.
- **table** indicates the target table of the bitmap scan.
- **index_list** indicates the index used by the bitmap scan.

Example

```
gaussdb=# explain(costs off)  select /*+ BitmapScan(t1 it1 it3)*/* from t1 where  (t1.c1 = 5 or t1.c2=6) or (t1.c3=3 or t1.c2=7);
                        QUERY PLAN
------------------------------------------------------------------
 Bitmap Heap Scan on t1
   Recheck Cond: ((c1 = 5) OR (c2 = 6) OR (c3 = 3) OR (c2 = 7))
   ->  BitmapOr
         ->  Bitmap Index Scan on it1
               Index Cond: (c1 = 5)
         ->  Bitmap Index Scan on it3
               Index Cond: (c2 = 6)
         ->  Bitmap Index Scan on it3
               Index Cond: (c3 = 3)
         ->  Bitmap Index Scan on it3
               Index Cond: (c2 = 7)
(11 rows)
```

NOTE

The path that meets the bitmap scan hint is selected from the existing index paths. Because the index path construction space is large and the optimizer prunes the paths, if any index path is not generated, the path cannot be constructed.

6.9.20 Hints for Inner Table Materialization During Join

Description

These hints materialize inner tables when specifying the inner tables to be joined.

Syntax

```
[no] materialize_inner([@queryblock] inner_table_list)
```

Parameters

- **no** indicates that the materialization of hints is not used.
- For details about **@queryblock**, see **Hint for Specifying the Query Block Where the Hint Is Located**. This parameter can be omitted, indicating that it takes effect in the current query block.
- **inner_table_list**: a list of inner tables to be materialized during the join operation. The value is a character string separated by spaces.

Example

Table **t1** is an inner table to be materialized, and the result of (t1 t2) is materialized as a joined inner table.

```
gaussdb=# explain (costs off) select /*+materialize_inner(t1) materialize_inner(t1 t2)*/ * from t1,t2,t3
where  t1.c3 = t2.c3 and t2.c2=t3.c2 and t1.c2=5;
                    QUERY PLAN
```



```
-----------------------------------------------------------
 Nested Loop
   Join Filter: (t2.c2 = t3.c2)
   ->  Seq Scan on t3
   ->  Materialize
         ->  Nested Loop
               Join Filter: (t1.c3 = t2.c3)
               ->  Seq Scan on t2
               ->  Materialize
                     ->  Bitmap Heap Scan on t1
                           Recheck Cond: (c2 = 5)
                           ->  Bitmap Index Scan on it3
                                 Index Cond: (c2 = 5)
(12 rows)
```

6.9.21 AGG Hint

Description

You can specify the AGG method when performing the AGG algorithm.

Syntax

```
use_hash_agg[(@queryblock)], use_sort_agg[(@queryblock)]
```

Parameters

For details about **@queryblock**, see **Hint for Specifying the Query Block Where the Hint Is Located**. This parameter can be omitted, indicating that it takes effect in the current query block. If it is not specified, the hint does not have parentheses (()).

Example

1. Use hash aggregation.

```
gaussdb=# explain (costs off) select c1 from t2 where c1 in( select /*+ use_hash_agg */ t1.c1 from t1,t3 where t1.c1=t3.c1 group by 1);
                 QUERY PLAN
-------------------------------------------------
 Hash Semi Join
   Hash Cond: (t2.c1 = t1.c1)
   ->  Seq Scan on t2
   ->  Hash
         ->  HashAggregate
               Group By Key: t1.c1
               ->  Hash Join
                     Hash Cond: (t1.c1 = t3.c1)
                     ->  Seq Scan on t1
                     ->  Hash
                           ->  Seq Scan on t3
(11 rows)
```

2. Use use_sort_agg for aggregation and then perform merge join.

```
gaussdb=# explain (costs off) select c1 from t2 where c1 in( select /*+ use_sort_agg */ t1.c1 from t1,t3 where t1.c1=t3.c1 group by 1);
                   QUERY PLAN
-----------------------------------------------------------
 Hash Semi Join
   Hash Cond: (t2.c1 = t1.c1)
   ->  Seq Scan on t2
   ->  Hash
         ->  Group
               Group By Key: t1.c1
               ->  Merge Join
```



```
                Merge Cond: (t1.c1 = t3.c1)
                ->  Index Only Scan using it1 on t1
                ->  Sort
                      Sort Key: t3.c1
                      ->  Seq Scan on t3
(12 rows)
```

6.9.22 Query Rewriting Hints

Description

The optimizer supports a series of query rewriting rules and can perform equivalent logical rewriting on SQL statements to generate a better execution plan. If you do not want to perform SQL statement rewriting or the optimizer rewriting results in plan changes, you need to use hints to control the rewriting rules so that the optimizer performs rewriting in the way you want. Currently, the database supports hint control for SQL statements in various scenarios, such as ANY/EXISTS sublinks, simple subqueries, ORDER BY removal, HAVING clause pushdown, and delayed aggregation. For details, see **Hint Usage Description**.

NOTICE

- Some query rewriting rules are controlled by both query rewriting hints and GUC parameters. Generally, query rewriting hints have a higher priority than GUC parameters. Rewriting rules controlled by GUC parameters are described in **Hint Usage Description**.
- Each query rewriting rule is controlled by a pair of mutually exclusive hints, for example: The subquery expansion rule is controlled by both EXPAND_SUBQUERY and NO_EXPAND_SUBQUERY. The EXPAND_SUBQUERY hint indicates that the rule can be used to rewrite SQL statements, and NO_EXPAND_SUBQUERY indicates that the rule cannot be used to rewrite SQL statements. If two mutually exclusive hints exist in the same query block, the first obtained hint prevails. For example, if **/*+ EXPAND_SUBQUERY NO_EXPAND_SUBQUERY */** is obtained, the EXPAND_SUBQUERY hint takes effect.
- Duplicate hints are allowed. However, for duplicate hints, only the first one is used. For other unused hints, an unused hint warning is reported. Take **/*+ EXPAND_SUBLINK EXPAND_SUBLINK */** as an example, database uses only the first EXPAND_SUBLINK hint. Therefore, the unused hint warning is still displayed.

Syntax

```
hintname[(@queryblock)]
```

Parameters

- **hintname** specifies the name of the hint that controls the query rewriting rule. For details about the supported query rewriting hints, see **Query rewriting hints**.
- For details about **@queryblock**, see **Hint for Specifying the Query Block Where the Hint Is Located**. This parameter can be omitted, indicating that it

takes effect in the current query block. If it is not specified, the hint does not have parentheses (()).

Query Rewriting Hint List

Table 6-3 Query rewriting hints

No.	Hint	Description
1	EXPAND_SUBLINK_HAVING	Sublinks in the HAVING clause can be pulled up.
2	NO_EXPAND_SUBLINK_HAVING	Sublinks in the HAVING clause cannot be pulled up.
3	EXPAND_SUBLINK	Sublinks of the ANY/EXISTS type can be pulled up.
4	NO_EXPAND_SUBLINK	Sublinks of the ANY/EXISTS type cannot be pulled up.
5	EXPAND_SUBLINK_TARGET	Sublinks in the TargetList can be pulled up.
6	NO_EXPAND_SUBLINK_TARGET	Sublinks in the TargetList cannot be pulled up.
7	USE_MAGIC_SET	When conditions are pushed down from the main query to the subquery, the join columns of the subquery are grouped and aggregated, and then the subquery is joined with the main query. This reduces repeated scanning of related sublinks and improves query efficiency.
8	NO_USE_MAGIC_SET	Conditions cannot be pushed down from the main query to the subquery. The subquery with an aggregate operator is joined with the main query in advance.
9	EXPAND_SUBLINK_UNIQUE_CHECK	Sublinks without aggregation can be pulled up if only one line is output for each condition. Sublinks with aggregation can be automatically pulled up.
10	NO_EXPAND_SUBLINK_UNIQUE_CHECK	Sublinks without aggregation cannot be pulled up.
11	NO_SUBLINK_DISABLE_REPLICATED	Sublinks can be pulled up in the fast query shipping plan or stream scenario with a replication table.
12	SUBLINK_DISABLE_REPLICATED	Sublinks cannot be pulled up in the fast query shipping plan or stream scenario with a replication table.

No.	Hint	Description
13	NO_SUBLINK_DISABLE_EXPR	Expressions in sublinks can be pulled up.
14	SUBLINK_DISABLE_EXPR	Expressions in sublinks cannot be pulled up.
15	ENABLE_SUBLINK_ENHANCED	Pulling up sublinks can be enhanced. Related or non-related sublinks such as OR expressions can be pulled up.
16	NO_ENABLE_SUBLINK_ENHANCED	Pulling up sublinks cannot be enhanced. Related or non-related sublinks such as OR expressions cannot be pulled up.
17	PARTIAL_PUSH	In the stream scenario, the gather operator can be added to listagg and arrayagg.
18	NO_PARTIAL_PUSH	In the stream scenario, the gather operator cannot be added to listagg and arrayagg.
19	REDUCE_ORDER_BY	Redundant ORDER BY can be reduced. If the outer query does not have sorting requirements on the inner query result, unnecessary ORDER BY can be reduced to improve the query efficiency.
20	NO_REDUCE_ORDER_BY	Unnecessary ORDER BY cannot be reduced.
21	REMOVE_NOT_NULL	The unnecessary NOT NULL condition can be deleted. When the column attribute is **NOT NULL**, the IS NOT NULL judgment in the query condition can be deleted.
22	NO_REMOVE_NOT_NULL	The IS NOT NULL conditions cannot be deleted.
23	LAZY_AGG	The subquery and outer query have the same GROUP BY condition. The two-layer aggregation operation may cause low query efficiency. The aggregation operation in the subquery can be eliminated to improve query efficiency.
24	NO_LAZY_AGG	The aggregation operation rules in the subquery cannot be eliminated.
25	EXPAND_SUBQUERY	The subquery is pulled up and joined with the upper layer to optimize the query efficiency.
26	NO_EXPAND_SUBQUERY	The subquery cannot be pulled up.
27	PUSHDOWN_HAVING	The HAVING condition expression can be pushed down.

No.	Hint	Description
28	NO_PUSHDOWN_HAVING	The HAVING expression cannot be pushed down.
29	INLIST_TO_JOIN	The inlist-to-join can be used to rewrite SQL statements.
30	NO_INLIST_TO_JOIN	The inlist-to-join cannot be used to rewrite SQL statements.
31	ROWNUM_PUSHDOWN	Row numbers can be pushed down.
32	NO_ROWNUM_PUSHDOWN	Row numbers cannot be pushed down.
33	WINDOWAGG_PUSHDOWN	The filter criterion of the window function in the parent query can be pushed down to the subquery.
34	NO_WINDOWAGG_PUSHDOWN	The filter criterion of the window function in the parent query cannot be pushed down to the subquery.

Preparations for Using Hints

To help you understand the application scenarios of hints, this document provides all application examples of query rewriting hints. For details, see **Hint Usage Description**. The table creation statements and environment preparation are as follows:

- Session settings:

```
SET client_encoding = 'UTF8';
CREATE SCHEMA rewrite_rule_test;
SET current_schema = rewrite_rule_test;
SET enable_codegen= off;
```

- Table creation statements:

```
CREATE TABLE rewrite_rule_hint_t1 (a INT, b INT, c INT, d INT);
CREATE TABLE rewrite_rule_hint_t2 (a INT, b INT, c INT, d INT);
CREATE TABLE rewrite_rule_hint_t3 (a INT, b INT, c INT, d INT);
CREATE TABLE rewrite_rule_hint_t4 (a INT NOT NULL, b INT, c INT, d INT);
CREATE TABLE rewrite_rule_hint_t5 (slot INTEGER NOT NULL,cid BIGINT NOT NULL,name
CHARACTER VARYING NOT NULL) WITH (ORIENTATION = row);
INSERT INTO rewrite_rule_hint_t5 (slot, cid, name) values(generate_series(1, 10),generate_series(1,
10),'records.storage.state');
ANALYZE rewrite_rule_hint_t5;
CREATE TABLE rewrite_rule_hint_customer (
    c_custkey INTEGER NOT NULL,
    c_name CHARACTER VARYING(25) NOT NULL,
    c_address CHARACTER VARYING(40) NOT NULL,
    c_nationkey INTEGER NOT NULL,
    c_phone CHARACTER(15) NOT NULL,
    c_acctbal NUMERIC(15, 2) NOT NULL,
    c_mktsegment CHARACTER(10) NOT NULL,
    c_comment CHARACTER VARYING(117) NOT NULL
);
CREATE TABLE rewrite_rule_hint_orders (
    o_orderkey INTEGER NOT NULL,
    o_custkey INTEGER NOT NULL,
    o_orderstatus CHARACTER(1) NOT NULL,
```



```
    o_totalprice NUMERIC(15, 2) NOT NULL,
    o_orderdate DATE NOT NULL,
    o_orderpriority CHARACTER(15) NOT NULL,
    o_clerk CHARACTER(15) NOT NULL,
    o_shippriority INTEGER NOT NULL,
    o_comment CHARACTER VARYING(79) NOT NULL
);
```

Hint Usage Description

1. EXPAND_SUBLINK_HAVING

   Sublinks in the HAVING clause can be pulled up. This scenario is also controlled by the GUC parameter **rewrite_rule**. If the hint of this rule is not used and the value of **rewrite_rule** is **enable_sublink_pullup_enhanced**, this rule also takes effect. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and NO_EXPAND_SUBLINK_HAVING are mutually exclusive.

   ```
   gaussdb=# EXPLAIN(costs off)SELECT /*+EXPAND_SUBLINK_HAVING*/ a,sum(b) AS value FROM
   rewrite_rule_hint_t1 GROUP BY a HAVING sum(a) >= (SELECT avg(b) FROM rewrite_rule_hint_t1)
   ORDER BY value DESC;
                            QUERY PLAN
   ------------------------------------------------------------------------
    Sort
      Sort Key: inner_subquery.value DESC
      InitPlan 1 (returns $0)
        ->  Aggregate
              ->  Seq Scan on rewrite_rule_hint_t1
      ->  Subquery Scan on inner_subquery
            ->  HashAggregate
                  Group By Key: rewrite_rule_test.rewrite_rule_hint_t1.a
                  Filter: ((sum(rewrite_rule_test.rewrite_rule_hint_t1.a))::numeric >= $0)
                  ->  Seq Scan on rewrite_rule_hint_t1
   (10 rows)
   ```

2. NO_EXPAND_SUBLINK_HAVING

   Sublinks in the HAVING clause cannot be pulled up. This scenario is also controlled by the GUC parameter **rewrite_rule**. If the hint of this rule is not used and the value of **rewrite_rule** is not **enable_sublink_pullup_enhanced**, this rule does not take effect. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and EXPAND_SUBLINK_HAVING are mutually exclusive.

   ```
   gaussdb=# EXPLAIN(costs off)SELECT /*+NO_EXPAND_SUBLINK_HAVING*/ a,sum(b) AS value FROM
   rewrite_rule_hint_t1 GROUP BY a HAVING sum(a) >= (SELECT avg(b) FROM rewrite_rule_hint_t1)
   ORDER BY value DESC;
                                  QUERY PLAN
   ---------------------------------------------------------------------------------
    Sort
      Sort Key: (sum(rewrite_rule_test.rewrite_rule_hint_t1.b)) DESC
      InitPlan 1 (returns $0)
        ->  Aggregate
              ->  Seq Scan on rewrite_rule_hint_t1
      ->  HashAggregate
            Group By Key: rewrite_rule_test.rewrite_rule_hint_t1.a
            Filter: ((sum(rewrite_rule_test.rewrite_rule_hint_t1.a))::numeric >= $0)
            ->  Seq Scan on rewrite_rule_hint_t1
   (9 rows)
   ```

3. EXPAND_SUBLINK

   Sublinks can be pulled up. It can control scenarios such as non-correlated sublinks of the [Not]Any type or related sublinks of the [Not]Exists type. The hint of this rule and the NO_EXPAND hint are mutually exclusive, and this hint has a higher priority than NO_EXPAND. This hint and NO_EXPAND_SUBLINK are mutually exclusive.


```
gaussdb=# EXPLAIN(costs off)SELECT * FROM rewrite_rule_hint_t1 WHERE a > ANY(SELECT /*
+EXPAND_SUBLINK*/ a FROM rewrite_rule_hint_t2) AND b > ANY (SELECT /*+EXPAND_SUBLINK*/a
FROM rewrite_rule_hint_t3);
                          QUERY PLAN
------------------------------------------------------------------------
 Nested Loop Semi Join
   Join Filter: (rewrite_rule_hint_t1.b > rewrite_rule_hint_t3.a)
   ->  Nested Loop Semi Join
         Join Filter: (rewrite_rule_hint_t1.a > rewrite_rule_hint_t2.a)
         ->  Seq Scan on rewrite_rule_hint_t1
         ->  Materialize
               ->  Seq Scan on rewrite_rule_hint_t2
   ->  Materialize
         ->  Seq Scan on rewrite_rule_hint_t3
(9 rows)
```

4. NO_EXPAND_SUBLINK

   Sublinks cannot be pulled up. It can control scenarios such as non-correlated sublinks of the [Not]Any type or related sublinks of the [Not]Exists type. The hint of this rule is equivalent to the NO_EXPAND hint. This hint and EXPAND_SUBLINK are mutually exclusive.

```
gaussdb=# EXPLAIN(costs off)SELECT * FROM rewrite_rule_hint_t1 WHERE a > ANY(SELECT /*
+NO_EXPAND_SUBLINK*/ a FROM rewrite_rule_hint_t2) AND b > ANY (SELECT /*
+EXPAND_SUBLINK*/a FROM rewrite_rule_hint_t3);
                    QUERY PLAN
-----------------------------------------------------
 Seq Scan on rewrite_rule_hint_t1
   Filter: ((NOT (hashed SubPlan 2)) AND (SubPlan 1))
   SubPlan 2
     ->  Seq Scan on rewrite_rule_hint_t3
   SubPlan 1
     ->  Materialize
           ->  Seq Scan on rewrite_rule_hint_t2
(7 rows)
```

5. EXPAND_SUBLINK_TARGET

   Sublinks in the TargetList can be pulled up. The hint of this rule and the NO_EXPAND hint are mutually exclusive, and this hint has a higher priority than NO_EXPAND. This scenario is also controlled by the GUC parameter **rewrite_rule**. If the hint of this rule is not used and the value of **rewrite_rule** is **intargetlist**, this rule also takes effect. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and NO_EXPAND_SUBLINK_TARGET are mutually exclusive.

```
gaussdb=# EXPLAIN(costs off)SELECT a,(SELECT /*+EXPAND_SUBLINK_TARGET*/ avg(b) FROM
rewrite_rule_hint_t1 WHERE rewrite_rule_hint_t1.b = rewrite_rule_hint_t2.b) FROM
rewrite_rule_hint_t2 WHERE rewrite_rule_hint_t2.a < 100 ORDER BY rewrite_rule_hint_t2.b;
                          QUERY PLAN
------------------------------------------------------------------------
 Sort
   Sort Key: rewrite_rule_hint_t2.b
   ->  Hash Left Join
         Hash Cond: (rewrite_rule_hint_t2.b = rewrite_rule_hint_t1.b)
         ->  Seq Scan on rewrite_rule_hint_t2
               Filter: (a < 100)
         ->  Hash
               ->  HashAggregate
                     Group By Key: rewrite_rule_hint_t1.b
                     ->  Seq Scan on rewrite_rule_hint_t1
(10 rows)
```

6. NO_EXPAND_SUBLINK_TARGET

   Sublinks in the TargetList cannot be pulled up. The hint of this rule is equivalent to the NO_EXPAND hint. This scenario is also controlled by the GUC parameter **rewrite_rule**. If the hint of this rule is not used and the value

of **rewrite_rule** is not **intargetlist**, this rule does not take effect. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and EXPAND_SUBLINK_TARGET are mutually exclusive.

```
gaussdb=# EXPLAIN(costs off)SELECT a,(SELECT /*+NO_EXPAND_SUBLINK_TARGET*/ avg(b) FROM
rewrite_rule_hint_t1 WHERE rewrite_rule_hint_t1.b = rewrite_rule_hint_t2.b) FROM
rewrite_rule_hint_t2 WHERE rewrite_rule_hint_t2.a < 100 ORDER BY rewrite_rule_hint_t2.b;
                        QUERY PLAN
------------------------------------------------------------
 Sort
   Sort Key: rewrite_rule_hint_t2.b
   ->  Seq Scan on rewrite_rule_hint_t2
         Filter: (a < 100)
         SubPlan 1
           ->  Aggregate
                 ->  Seq Scan on rewrite_rule_hint_t1
                       Filter: (b = rewrite_rule_hint_t2.b)
(8 rows)
```

7. USE_MAGIC_SET

   Conditions are pushed down from the main query to the subquery. In this case, the join columns of the subquery are grouped and aggregated, and then the subquery is joined with the main query. This reduces repeated scanning of related sublinks and improves query efficiency. This scenario is also controlled by the GUC parameter **rewrite_rule**. If the hint of this rule is not used and the value of **rewrite_rule** is **magicset**, this rule also takes effect. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and NO_USE_MAGIC_SET are mutually exclusive.

```
gaussdb=# EXPLAIN(costs off) SELECT rewrite_rule_hint_t1 FROM rewrite_rule_hint_t1 WHERE
rewrite_rule_hint_t1.b = 10 AND rewrite_rule_hint_t1.c < (SELECT /*+USE_MAGIC_SET*/ sum(c) FROM
rewrite_rule_hint_t2 WHERE rewrite_rule_hint_t1.a = rewrite_rule_hint_t2.a);
                                    QUERY PLAN
-----------------------------------------------------------------------------------------------
 Hash Join
   Hash Cond: (rewrite_rule_hint_t2.a = rewrite_rule_test.rewrite_rule_hint_t1.a)
   Join Filter: (rewrite_rule_test.rewrite_rule_hint_t1.c < (sum(rewrite_rule_hint_t2.c)))
   ->  HashAggregate
         Group By Key: rewrite_rule_hint_t2.a
         ->  Hash Join
               Hash Cond: (rewrite_rule_hint_t2.a = rewrite_rule_test.rewrite_rule_hint_t1.a)
               ->  Seq Scan on rewrite_rule_hint_t2
               ->  Hash
                     ->  HashAggregate
                           Group By Key: rewrite_rule_test.rewrite_rule_hint_t1.a
                           ->  Seq Scan on rewrite_rule_hint_t1
                                 Filter: (b = 10)
   ->  Hash
         ->  Seq Scan on rewrite_rule_hint_t1
               Filter: (b = 10)
(16 rows)
```

8. NO_USE_MAGIC_SET

   Conditions cannot be pushed down from the main query to the subquery. The subquery with an aggregate operator is joined with the main query in advance. This scenario is also controlled by the GUC parameter **rewrite_rule**. If the hint of this rule is not used and the value of **rewrite_rule** is not **magicset**, this rule also does not take effect. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and USE_MAGIC_SET are mutually exclusive.

```
gaussdb=# EXPLAIN(costs off) SELECT rewrite_rule_hint_t1 FROM rewrite_rule_hint_t1 WHERE
rewrite_rule_hint_t1.b = 10 AND rewrite_rule_hint_t1.c < (SELECT /*+NO_USE_MAGIC_SET*/ sum(c)
```



```
FROM rewrite_rule_hint_t2 WHERE rewrite_rule_hint_t1.a = rewrite_rule_hint_t2.a);
                            QUERY PLAN
------------------------------------------------------------------------------
 Hash Join
   Hash Cond: (rewrite_rule_hint_t2.a = rewrite_rule_hint_t1.a)
   Join Filter: (rewrite_rule_hint_t1.c < (sum(rewrite_rule_hint_t2.c)))
   ->  HashAggregate
         Group By Key: rewrite_rule_hint_t2.a
         ->  Seq Scan on rewrite_rule_hint_t2
   ->  Hash
         ->  Seq Scan on rewrite_rule_hint_t1
               Filter: (b = 10)
(9 rows)
```

9. EXPAND_SUBLINK_UNIQUE_CHECK

   Sublinks without aggregation are pulled up. To pull up sublinks, ensure that only one line is output for each condition. Sublinks with aggregation can be automatically pulled up. This scenario is also controlled by the GUC parameter **rewrite_rule**. If the hint of this rule is not used and the value of **rewrite_rule** is **uniquecheck**, this rule also takes effect. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and NO_EXPAND_SUBLINK_UNIQUE_CHECK are mutually exclusive.

```
gaussdb=# EXPLAIN(costs off)SELECT rewrite_rule_hint_t1.a FROM rewrite_rule_hint_t1 WHERE
rewrite_rule_hint_t1.a = (SELECT /*+EXPAND_SUBLINK_UNIQUE_CHECK*/ rewrite_rule_hint_t2.a FROM
rewrite_rule_hint_t2 WHERE rewrite_rule_hint_t1.a = rewrite_rule_hint_t2.b);
                              QUERY PLAN
-------------------------------------------------------------------------------
 Hash Join
   Hash Cond: (rewrite_rule_hint_t1.a = subquery."?column?")
   ->  Seq Scan on rewrite_rule_hint_t1
   ->  Hash
         ->  Subquery Scan on subquery
               ->  HashAggregate
                     Group By Key: rewrite_rule_hint_t2.b
                     Filter: (rewrite_rule_hint_t2.b = rewrite_rule_hint_t2.a)
                     Unique Check Required
                     ->  Seq Scan on rewrite_rule_hint_t2
(10 rows)
```

10. NO_EXPAND_SUBLINK_UNIQUE_CHECK

    Sublinks without aggregation cannot be pulled up. This scenario is also controlled by the GUC parameter **rewrite_rule**. If the hint of this rule is not used and the value of **rewrite_rule** is not **uniquecheck**, this rule does not take effect. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and EXPAND_SUBLINK_UNIQUE_CHECK are mutually exclusive.

```
gaussdb=# EXPLAIN(costs off)SELECT rewrite_rule_hint_t1.a FROM rewrite_rule_hint_t1 WHERE
rewrite_rule_hint_t1.a = (SELECT /*+NO_EXPAND_SUBLINK_UNIQUE_CHECK*/ rewrite_rule_hint_t2.a
FROM rewrite_rule_hint_t2 WHERE rewrite_rule_hint_t1.a = rewrite_rule_hint_t2.b);
                 QUERY PLAN
-------------------------------------------------
 Seq Scan on rewrite_rule_hint_t1
   Filter: (a = (SubPlan 1))
   SubPlan 1
     ->  Seq Scan on rewrite_rule_hint_t2
           Filter: (rewrite_rule_hint_t1.a = b)
(5 rows)
```

11. NO_SUBLINK_DISABLE_REPLICATED

    Sublinks can be pulled up in the fast query shipping plan or stream scenario with a replication table. This scenario is also controlled by the GUC parameter **rewrite_rule**. If the hint of this rule is not used and the value of **rewrite_rule**

is not **disablerep**, this rule also takes effect. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and SUBLINK_DISABLE_REPLICATED are mutually exclusive.

```
gaussdb=# EXPLAIN(costs off)SELECT * FROM rewrite_rule_hint_t1 WHERE (0 =(SELECT /*
+NO_SUBLINK_DISABLE_REPLICATED*/ count(*) FROM rewrite_rule_hint_t2 WHERE
rewrite_rule_hint_t2.a = rewrite_rule_hint_t1.a) OR NOT EXISTS(SELECT /*
+NO_SUBLINK_DISABLE_REPLICATED*/1 FROM rewrite_rule_hint_t3 WHERE rewrite_rule_hint_t3.b =
rewrite_rule_hint_t1.b));
                                        QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
----
 Hash Left Join
   Hash Cond: (rewrite_rule_hint_t1.b = rewrite_rule_hint_t3.b)
   Filter: (((subquery."?column?" IS NOT NULL) AND (0 = COALESCE(subquery.count, 0))) OR
(rewrite_rule_hint_t3.b IS NULL))
   ->  Hash Left Join
         Hash Cond: (rewrite_rule_hint_t1.a = subquery."?column?")
         ->  Seq Scan on rewrite_rule_hint_t1
         ->  Hash
               ->  Subquery Scan on subquery
                     ->  HashAggregate
                           Group By Key: rewrite_rule_hint_t2.a
                           ->  Seq Scan on rewrite_rule_hint_t2
   ->  Hash
         ->  HashAggregate
               Group By Key: rewrite_rule_hint_t3.b
               ->  Seq Scan on rewrite_rule_hint_t3
(15 rows)
```

12. SUBLINK_DISABLE_REPLICATED

    Sublinks cannot be pulled up in the fast query shipping plan or stream scenario with a replication table. This scenario is also controlled by the GUC parameter **rewrite_rule**. If the hint of this rule is not used and the value of **rewrite_rule** is **disablerep**, this rule does not take effect. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and NO_SUBLINK_DISABLE_REPLICATED are mutually exclusive.

    ```
    gaussdb=# EXPLAIN(costs off)SELECT * FROM rewrite_rule_hint_t1 WHERE (0 =(SELECT /*
    +SUBLINK_DISABLE_REPLICATED*/ count(*) FROM rewrite_rule_hint_t2 WHERE rewrite_rule_hint_t2.a
    = rewrite_rule_hint_t1.a) OR NOT EXISTS(SELECT /*+NO_SUBLINK_DISABLE_REPLICATED*/1 FROM
    rewrite_rule_hint_t3 WHERE rewrite_rule_hint_t3.b = rewrite_rule_hint_t1.b));
                                        QUERY PLAN
    ------------------------------------------------------------------------------------
     Seq Scan on rewrite_rule_hint_t1
       Filter: ((0 = (SubPlan 1)) OR (NOT (alternatives: SubPlan 2 or hashed SubPlan 3)))
       SubPlan 1
         ->  Aggregate
               ->  Seq Scan on rewrite_rule_hint_t2
                     Filter: (a = rewrite_rule_hint_t1.a)
       SubPlan 2
         ->  Seq Scan on rewrite_rule_hint_t3
               Filter: (b = rewrite_rule_hint_t1.b)
       SubPlan 3
         ->  Seq Scan on rewrite_rule_hint_t3
    (11 rows)
    ```

13. NO_SUBLINK_DISABLE_EXPR

    Expressions in sublinks can be pulled up. This scenario is also controlled by the GUC parameter **rewrite_rule**. If the hint of this rule is not used and the value of **rewrite_rule** is not **disable_pullup_expr_sublink**, this rule also takes effect. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and SUBLINK_DISABLE_EXPR are mutually exclusive.


```
gaussdb=# EXPLAIN(costs off)SELECT a FROM rewrite_rule_hint_t1 WHERE rewrite_rule_hint_t1.b =
(SELECT /*+NO_SUBLINK_DISABLE_EXPR*/ max(b) FROM rewrite_rule_hint_t2 WHERE
rewrite_rule_hint_t2.a = rewrite_rule_hint_t1.a);
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------
 Hash Join
   Hash Cond: ((rewrite_rule_hint_t1.a = subquery."?column?") AND (rewrite_rule_hint_t1.b =
subquery.max))
   ->  Seq Scan on rewrite_rule_hint_t1
   ->  Hash
         ->  Subquery Scan on subquery
               ->  HashAggregate
                     Group By Key: rewrite_rule_hint_t2.a
                     ->  Seq Scan on rewrite_rule_hint_t2
(8 rows)
```

14. SUBLINK_DISABLE_EXPR

    Expressions in sublinks cannot be pulled up. This scenario is also controlled by the GUC parameter **rewrite_rule**. If the hint of this rule is not used and the value of **rewrite_rule** is **disable_pullup_expr_sublink**, this rule does not take effect. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and NO_SUBLINK_DISABLE_EXPR are mutually exclusive.

```
gaussdb=# EXPLAIN(costs off)SELECT a FROM rewrite_rule_hint_t1 WHERE rewrite_rule_hint_t1.b =
(SELECT /*+SUBLINK_DISABLE_EXPR*/ max(b) FROM rewrite_rule_hint_t2 WHERE
rewrite_rule_hint_t2.a = rewrite_rule_hint_t1.a);
                      QUERY PLAN
---------------------------------------------------
 Seq Scan on rewrite_rule_hint_t1
   Filter: (b = (SubPlan 1))
   SubPlan 1
     ->  Aggregate
           ->  Seq Scan on rewrite_rule_hint_t2
                 Filter: (a = rewrite_rule_hint_t1.a)
(6 rows)
```

15. ENABLE_SUBLINK_ENHANCED

    The sublink pullup can be enhanced. This scenario is also controlled by the GUC parameter **rewrite_rule**. If the hint of this rule is not used and the value of **rewrite_rule** is **enable_sublink_pullup_enhanced**, this rule also takes effect. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and NO_ENABLE_SUBLINK_ENHANCED are mutually exclusive.

```
gaussdb=# EXPLAIN (costs off)SELECT cntrycode,count(*) AS numcust,sum(c_acctbal) AS totacctbal
FROM (SELECT substring(c_phone from 1 for 2) AS cntrycode,c_acctbal FROM
rewrite_rule_hint_customer WHERE substring(c_phone from 1 for 2) IN ('22', '25', '26', '14', '18', '30',
'17')AND c_acctbal > (SELECT /*+ENABLE_SUBLINK_ENHANCED*/ avg(c_acctbal) FROM
rewrite_rule_hint_customer WHERE c_acctbal > 0.00 AND substring(c_phone from 1 for 2) IN ('22',
'25', '26', '14', '18', '30', '17')) AND NOT EXISTS (SELECT * FROM rewrite_rule_hint_orders WHERE
o_custkey = c_custkey)) AS custsale GROUP BY cntrycode ORDER BY cntrycode;
                                                               QUERY
PLAN
-----------------------------------------------------------------------------------------------------------------
----------------------------------------
 GroupAggregate
   Group By Key: ("substring"((rewrite_rule_test.rewrite_rule_hint_customer.c_phone)::text, 1, 2))
   ->  Sort
         Sort Key: ("substring"((rewrite_rule_test.rewrite_rule_hint_customer.c_phone)::text, 1, 2))
         ->  Hash Right Anti Join
               Hash Cond: (rewrite_rule_hint_orders.o_custkey =
rewrite_rule_test.rewrite_rule_hint_customer.c_custkey)
               ->  Seq Scan on rewrite_rule_hint_orders
               ->  Hash
                     ->  Nested Loop
                           Join Filter: (rewrite_rule_test.rewrite_rule_hint_customer.c_acctbal >
```

```
(avg(rewrite_rule_test.rewrite_rule_hint_customer.c_acctbal)))
                    ->  Aggregate
                          ->  Seq Scan on rewrite_rule_hint_customer
                                Filter: ((c_acctbal > 0.00) AND ("substring"((c_phone)::text, 1, 2) = ANY
('{22,25,26,14,18,30,17}'::text[])))
                    ->  Seq Scan on rewrite_rule_hint_customer
                          Filter: ("substring"((c_phone)::text, 1, 2) = ANY ('{22,25,26,14,18,30,17}'::text[]))
(15 rows)
```

16. NO_ENABLE_SUBLINK_ENHANCED

    The sublink pullup cannot be enhanced. This scenario is also controlled by the GUC parameter **rewrite_rule**. If the hint of this rule is not used and the value of **rewrite_rule** is not **enable_sublink_pullup_enhanced**, this rule does not take effect. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and ENABLE_SUBLINK_ENHANCED are mutually exclusive.

```
gaussdb=# EXPLAIN (costs off)SELECT cntrycode,count(*) AS numcust,sum(c_acctbal) AS totacctbal
FROM (SELECT substring(c_phone from 1 for 2) AS cntrycode,c_acctbal FROM
rewrite_rule_hint_customer WHERE substring(c_phone from 1 for 2) IN ('22', '25', '26', '14', '18', '30',
'17')AND c_acctbal > (SELECT /*+NO_ENABLE_SUBLINK_ENHANCED*/ avg(c_acctbal) FROM
rewrite_rule_hint_customer WHERE c_acctbal > 0.00 AND substring(c_phone from 1 for 2) IN ('22',
'25', '26', '14', '18', '30', '17')) AND NOT EXISTS (SELECT * FROM rewrite_rule_hint_orders WHERE
o_custkey = c_custkey)) AS custsale GROUP BY cntrycode ORDER BY cntrycode;
                                                  QUERY PLAN
--------------------------------------------------------------------------------------------------------------------
-----------------
 GroupAggregate
   Group By Key: ("substring"((rewrite_rule_test.rewrite_rule_hint_customer.c_phone)::text, 1, 2))
   InitPlan 1 (returns $0)
     ->  Aggregate
           ->  Seq Scan on rewrite_rule_hint_customer
                 Filter: ((c_acctbal > 0.00) AND ("substring"((c_phone)::text, 1, 2) = ANY
('{22,25,26,14,18,30,17}'::text[])))
   ->  Sort
         Sort Key: ("substring"((rewrite_rule_test.rewrite_rule_hint_customer.c_phone)::text, 1, 2))
         ->  Hash Right Anti Join
               Hash Cond: (rewrite_rule_hint_orders.o_custkey =
rewrite_rule_test.rewrite_rule_hint_customer.c_custkey)
               ->  Seq Scan on rewrite_rule_hint_orders
               ->  Hash
                     ->  Seq Scan on rewrite_rule_hint_customer
                           Filter: ((c_acctbal > $0) AND ("substring"((c_phone)::text, 1, 2) = ANY
('{22,25,26,14,18,30,17}'::text[])))
(14 rows)
```

17. PARTIAL_PUSH

    In the stream scenario, the gather operator can be added to listagg and arrayagg. This scenario is also controlled by the GUC parameter **rewrite_rule**. If the hint of this rule is not used and the value of **rewrite_rule** is **partialpush**, this rule also takes effect. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and NO_PARTIAL_PUSH are mutually exclusive.

```
gaussdb=# SET rewrite_rule='intargetlist';
SET query_dop = 10;
SET enable_force_smp = on;
EXPLAIN (costs off)SELECT /*+PARTIAL_PUSH*/listagg((SELECT b FROM rewrite_rule_hint_t2 WHERE
rewrite_rule_hint_t2.b = rewrite_rule_hint_t1.b ORDER BY rewrite_rule_hint_t2.c limit 1), ',') WITHIN
GROUP(ORDER BY rewrite_rule_hint_t1.b) FROM rewrite_rule_hint_t1 ORDER BY 1;
                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------
 Sort
   Sort Key: (listagg(subquery.b, ','::text ) WITHIN GROUP ( ORDER BY rewrite_rule_hint_t1.b))
   ->  Aggregate
         ->  Streaming(type: LOCAL GATHER dop: 1/10)
               ->  Hash Right Join
```

```
                Hash Cond: (subquery."?column?" = rewrite_rule_hint_t1.b)
                ->  Streaming(type: BROADCAST dop: 10/10)
                     ->  Subquery Scan on subquery
                          Filter: (subquery."?column?" OPERATOR(pg_catalog.=) 1::bigint)
                          ->  WindowAgg
                               ->  Sort
                                    Sort Key: rewrite_rule_hint_t2.b, rewrite_rule_hint_t2.c
                                    ->  Streaming(type: LOCAL REDISTRIBUTE dop: 10/10)
                                         ->  Seq Scan on rewrite_rule_hint_t2
                ->  Hash
                     ->  Seq Scan on rewrite_rule_hint_t1
(16 rows)
```

18. NO_PARTIAL_PUSH

    In the stream scenario, the gather operator cannot be added to listagg and arrayagg. This scenario is also controlled by the GUC parameter **rewrite_rule**. If the hint of this rule is not used and the value of **rewrite_rule** is not **partialpush**, this rule does not take effect. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and PARTIAL_PUSH are mutually exclusive.

```
gaussdb=# SET rewrite_rule='intargetlist';
SET query_dop = 10;
SET enable_force_smp = on;
EXPLAIN (costs off)SELECT /*+NO_PARTIAL_PUSH*/listagg((SELECT b FROM rewrite_rule_hint_t2
WHERE rewrite_rule_hint_t2.b = rewrite_rule_hint_t1.b ORDER BY rewrite_rule_hint_t2.c limit 1), ',')
WITHIN GROUP(ORDER BY rewrite_rule_hint_t1.b) FROM rewrite_rule_hint_t1 ORDER BY 1;
                                    QUERY PLAN
-----------------------------------------------------------------------------------------------
 Sort
   Sort Key: (listagg(subquery.b, ','::text ) WITHIN GROUP ( ORDER BY rewrite_rule_hint_t1.b))
   ->  Aggregate
        ->  Hash Right Join
             Hash Cond: (subquery."?column?" = rewrite_rule_hint_t1.b)
             ->  Subquery Scan on subquery
                  Filter: (subquery."?column?" OPERATOR(pg_catalog.=) 1::bigint)
                  ->  WindowAgg
                       ->  Sort
                            Sort Key: rewrite_rule_hint_t2.b, rewrite_rule_hint_t2.c
                            ->  Seq Scan on rewrite_rule_hint_t2
             ->  Hash
                  ->  Seq Scan on rewrite_rule_hint_t1
(13 rows)
```

19. REDUCE_ORDER_BY

    Unnecessary ORDER BY can be reduced. If the outer query does not have sorting requirements on the inner query result, unnecessary ORDER BY can be reduced to improve the query efficiency. This hint and NO_REDUCE_ORDER_BY are mutually exclusive.

```
gaussdb=# EXPLAIN(costs off)SELECT * FROM rewrite_rule_hint_t1,(SELECT /*+REDUCE_ORDER_BY*/ *
FROM rewrite_rule_hint_t2 ORDER BY a DESC);
               QUERY PLAN
------------------------------------------------
 Nested Loop
   ->  Seq Scan on rewrite_rule_hint_t1
   ->  Materialize
        ->  Seq Scan on rewrite_rule_hint_t2
(4 rows)
```

20. NO_REDUCE_ORDER_BY

    Unnecessary ORDER BY cannot be reduced. This hint and REDUCE_ORDER_BY are mutually exclusive.

```
gaussdb=# EXPLAIN(costs off)SELECT * FROM rewrite_rule_hint_t1,(SELECT /*
+NO_REDUCE_ORDER_BY*/ * FROM rewrite_rule_hint_t2 ORDER BY a DESC);
                   QUERY PLAN
-----------------------------------------------------
```



```
 Nested Loop
   ->  Seq Scan on rewrite_rule_hint_t1
   ->  Materialize
         ->  Sort
               Sort Key: rewrite_rule_hint_t2.a DESC
               ->  Seq Scan on rewrite_rule_hint_t2
(6 rows)
```

21. REMOVE_NOT_NULL

    Unnecessary IS NOT NULL conditions can be deleted. When the column attribute is **NOT NULL**, the IS NOT NULL judgment in the query condition can be deleted. This scenario is also controlled by the GUC parameter **enable_constraint_optimization**. If the hint of this rule is not used and the value of **enable_constraint_optimization** is **on**, this rule also takes effect. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and NO_REMOVE_NOT_NULL are mutually exclusive.

    ```
    gaussdb=# EXPLAIN(costs off)SELECT /*+REMOVE_NOT_NULL*/ * FROM rewrite_rule_hint_t4 WHERE
    b > 10 OR a IS NOT NULL;
              QUERY PLAN
    ----------------------------------
     Seq Scan on rewrite_rule_hint_t4
    (1 row)
    ```

22. NO_REMOVE_NOT_NULL

    Unnecessary IS NOT NULL conditions cannot be deleted. This scenario is also controlled by the GUC parameter **enable_constraint_optimization**. If the hint of this rule is not used and the value of **enable_constraint_optimization** is **off**, this rule does not take effect. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and REMOVE_NOT_NULL are mutually exclusive.

    ```
    gaussdb=# EXPLAIN(costs off)SELECT /*+NO_REMOVE_NOT_NULL*/ * FROM rewrite_rule_hint_t4
    WHERE b > 10 OR a IS NOT NULL;
                QUERY PLAN
    ----------------------------------------
     Seq Scan on rewrite_rule_hint_t4
       Filter: ((b > 10) OR (a IS NOT NULL))
    (2 rows)
    ```

23. LAZY_AGG

    The subquery and outer query have the same GROUP BY condition. The two-layer aggregation operation may cause low query efficiency. The aggregation operation in the subquery can be eliminated to improve query efficiency. This scenario is also controlled by the GUC parameter **rewrite_rule**. If the hint of this rule is not used and the value of **rewrite_rule** is **lazyagg**, this rule also takes effect. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and NO_LAZY_AGG are mutually exclusive.

    ```
    gaussdb=# EXPLAIN(costs off)SELECT rewrite_rule_hint_t1.b,sum(cc) FROM (SELECT /*+LAZY_AGG*/
    b,sum(c) AS cc FROM rewrite_rule_hint_t2 GROUP BY b) s1,rewrite_rule_hint_t1 WHERE s1.b =
    rewrite_rule_hint_t1.b GROUP BY rewrite_rule_hint_t1.b ORDER BY 1,2;
                                  QUERY PLAN
    ------------------------------------------------------------------------------
     Sort
       Sort Key: rewrite_rule_hint_t1.b, (sum((rewrite_rule_hint_t2.c)::bigint))
       ->  HashAggregate
             Group By Key: rewrite_rule_hint_t1.b
             ->  Hash Join
                   Hash Cond: (rewrite_rule_hint_t2.b = rewrite_rule_hint_t1.b)
                   ->  Seq Scan on rewrite_rule_hint_t2
                   ->  Hash
    ```



```
                ->  Seq Scan on rewrite_rule_hint_t1
(9 rows)
```

24. NO_LAZY_AGG

    The aggregation operation rules in the subquery cannot be eliminated. This scenario is also controlled by the GUC parameter **rewrite_rule**. If the hint of this rule is not used and the value of **rewrite_rule** is not **lazyagg**, this rule does not take effect. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and LAZY_AGG are mutually exclusive.

```
gaussdb=# EXPLAIN(costs off)SELECT rewrite_rule_hint_t1.b,sum(cc) FROM (SELECT /*
+NO_LAZY_AGG*/b,sum(c) AS cc FROM rewrite_rule_hint_t2 GROUP BY b) s1,rewrite_rule_hint_t1
WHERE s1.b = rewrite_rule_hint_t1.b GROUP BY rewrite_rule_hint_t1.b ORDER BY 1,2;
                        QUERY PLAN
------------------------------------------------------------------------
 Sort
   Sort Key: rewrite_rule_hint_t1.b, (sum(s1.cc))
   ->  HashAggregate
         Group By Key: rewrite_rule_hint_t1.b
         ->  Hash Join
               Hash Cond: (rewrite_rule_hint_t1.b = s1.b)
               ->  Seq Scan on rewrite_rule_hint_t1
               ->  Hash
                     ->  Subquery Scan on s1
                           ->  HashAggregate
                                 Group By Key: rewrite_rule_hint_t2.b
                                 ->  Seq Scan on rewrite_rule_hint_t2
(12 rows)
```

25. EXPAND_SUBQUERY

    The subquery is pulled up and joined with the upper layer to optimize the query efficiency. The hint of this rule and the NO_EXPAND hint are mutually exclusive. When both this hint and NO_EXPAND are used, this hint has a higher priority. This hint and NO_EXPAND_SUBQUERY are mutually exclusive.

```
gaussdb=# EXPLAIN(costs off) SELECT * FROM rewrite_rule_hint_t1,(SELECT /*+EXPAND_SUBQUERY*/
* FROM rewrite_rule_hint_t2 WHERE a > 1) tt WHERE rewrite_rule_hint_t1.a = tt.a;
                      QUERY PLAN
-----------------------------------------------------------------
 Hash Join
   Hash Cond: (rewrite_rule_hint_t1.a = rewrite_rule_hint_t2.a)
   ->  Seq Scan on rewrite_rule_hint_t1
         Filter: (a > 1)
   ->  Hash
         ->  Seq Scan on rewrite_rule_hint_t2
               Filter: (a > 1)
(7 rows)
```

26. NO_EXPAND_SUBQUERY

    The subquery cannot be pulled up. This hint is equivalent to the NO_EXPAND hint, but this hint has a higher priority. This hint and EXPAND_SUBQUERY are mutually exclusive.

```
gaussdb=# EXPLAIN(costs off) SELECT * FROM rewrite_rule_hint_t1,(SELECT /*
+NO_EXPAND_SUBQUERY*/ * FROM rewrite_rule_hint_t2 WHERE a > 1) tt WHERE
rewrite_rule_hint_t1.a = tt.a;
                      QUERY PLAN
-----------------------------------------------------------------
 Hash Join
   Hash Cond: (rewrite_rule_hint_t1.a = rewrite_rule_hint_t2.a)
   ->  Seq Scan on rewrite_rule_hint_t1
   ->  Hash
         ->  Seq Scan on rewrite_rule_hint_t2
               Filter: (a > 1)
(6 rows)
```


27. PUSHDOWN_HAVING

    The HAVING condition expression can be pushed down. This hint and NO_PUSHDOWN_HAVING are mutually exclusive.

```
gaussdb=# EXPLAIN(costs off)SELECT /*+PUSHDOWN_HAVING*/ sum(a),b,c FROM
rewrite_rule_hint_t1 WHERE b > 0 GROUP BY b,c HAVING sum(a) > 100 AND c > 0;
              QUERY PLAN
---------------------------------------
 HashAggregate
   Group By Key: b, c
   Filter: (sum(a) > 100)
   ->  Seq Scan on rewrite_rule_hint_t1
         Filter: ((b > 0) AND (c > 0))
(5 rows)
```

28. NO_PUSHDOWN_HAVING

    The HAVING expression cannot be pushed down. This hint and PUSHDOWN_HAVING are mutually exclusive.

```
gaussdb=# EXPLAIN(costs off)SELECT /*+NO_PUSHDOWN_HAVING*/ sum(a),b,c FROM
rewrite_rule_hint_t1 WHERE b > 0 GROUP BY b,c HAVING sum(a) > 100 AND c > 0;
              QUERY PLAN
---------------------------------------
 HashAggregate
   Group By Key: b, c
   Filter: ((sum(a) > 100) AND (c > 0))
   ->  Seq Scan on rewrite_rule_hint_t1
         Filter: (b > 0)
(5 rows)
```

29. INLIST_TO_JOIN

    The inlist-to-join can be used to rewrite SQL statements. This scenario is also controlled by the GUC parameter **qrw_inlist2join_optmode**. If the hint of this rule is not used and the value of **qrw_inlist2join_optmode** is **rule_base**, this rule takes effect. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and NO_INLIST_TO_JOIN are mutually exclusive.

```
gaussdb=# EXPLAIN (costs off)SELECT * FROM rewrite_rule_hint_t5 WHERE slot = '5' AND (name) IN
(SELECT /*+INLIST_TO_JOIN*/ name FROM rewrite_rule_hint_t5 WHERE slot = '5'AND cid IN
(5,1000,1001,1002,1003,1004,1005,1006,1007,2000,4000,10781986,10880002)LIMIT 50);
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
-----
 Nested Loop Semi Join
   Join Filter: ((rewrite_rule_test.rewrite_rule_hint_t5.name)::text =
(rewrite_rule_test.rewrite_rule_hint_t5.name)::text)
   ->  Seq Scan on rewrite_rule_hint_t5
         Filter: (slot = 5)
   ->  Limit
         ->  Hash Right Semi Join
               Hash Cond: ("*VALUES*".column1 = rewrite_rule_test.rewrite_rule_hint_t5.cid)
               ->  Values Scan on "*VALUES*"
               ->  Hash
                     ->  Seq Scan on rewrite_rule_hint_t5
                           Filter: (slot = 5)
(11 rows)
```


NOTE

INLIST_TO_JOIN[(@queryblock threshold)]: supports no parameter or any integer greater than or equal to 0. It is compatible with the value of the GUC parameter **qrw_inlist2join_optmode**. The default value is recommended.

Parameter description:

- **threshold**: specifies the query rewriting threshold. This parameter is optional. For details, see the value range part.

Default value: **1** (Optional. If it is not set, the default value is used.)

Value range:

- **0**: cost_base
- **1**: rule_base
- Any other positive integer: specifies the query rewriting threshold. If the number of elements in the list is greater than the threshold, the inlist2join query rewriting is performed.

30. NO_INLIST_TO_JOIN

The inlist-to-join cannot be used to rewrite SQL statements. This scenario is also controlled by the GUC parameter **qrw_inlist2join_optmode**. If the hint of this rule is not used and the value of **qrw_inlist2join_optmode** is **disable**, this rule does not take effect. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and INLIST_TO_JOIN are mutually exclusive.

```
gaussdb=# EXPLAIN (costs off)SELECT * FROM rewrite_rule_hint_t5 WHERE slot = '5' AND (name) IN
(SELECT /*+NO_INLIST_TO_JOIN*/ name FROM rewrite_rule_hint_t5 WHERE slot = '5'AND cid IN
(5,1000,1001,1002,1003,1004,1005,1006,1007,2000,4000,10781986,10880002)LIMIT 50);
                                                    QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
-------------------
 Nested Loop Semi Join
   Join Filter: ((rewrite_rule_test.rewrite_rule_hint_t5.name)::text =
(rewrite_rule_test.rewrite_rule_hint_t5.name)::text)
   ->  Seq Scan on rewrite_rule_hint_t5
         Filter: (slot = 5)
   ->  Limit
         ->  Seq Scan on rewrite_rule_hint_t5
               Filter: ((slot = 5) AND (cid = ANY
('{5,1000,1001,1002,1003,1004,1005,1006,1007,2000,4000,10781986,10880002}'::bigint[])))
(7 rows)
```

31. ROWNUM_PUSHDOWN

Row numbers can be pushed down. This scenario is also controlled by the GUC parameter **rewrite_rule**. If the hint of this rule is not used and the value of **rewrite_rule** is **disable_rownum_pushdown**, pushdown is allowed. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and NO_ROWNUM_PUSHDOWN are mutually exclusive.

```
gaussdb=# EXPLAIN (costs off) SELECT * FROM (SELECT /*+ROWNUM_PUSHDOWN*/rownum rn, a
FROM rewrite_rule_hint_t1) WHERE rn BETWEEN 5 AND 10;
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Subquery Scan on __unnamed_subquery__
   Filter: ((__unnamed_subquery__.rn >= 5) AND (__unnamed_subquery__.rn <= 10))
   ->  Rownum
         StopKey: (ROWNUM <= 10)
         ->  Seq Scan on rewrite_rule_hint_t1
(5 rows)
```

32. NO_ROWNUM_PUSHDOWN

Row numbers cannot be pushed down. This scenario is also controlled by the GUC parameter **rewrite_rule**. If the hint of this rule is not used and the value

of **rewrite_rule** is **disable_rownum_pushdown**, pushdown is not allowed. However, when both the hint and GUC parameter of this rule are used, the hint precedes the GUC parameter. This hint and ROWNUM_PUSHDOWN are mutually exclusive.

```
gaussdb=# EXPLAIN (costs off) SELECT * FROM (SELECT /*+NO_ROWNUM_PUSHDOWN*/rownum rn,
a FROM rewrite_rule_hint_t1) WHERE rn BETWEEN 5 AND 10;
                              QUERY PLAN
-------------------------------------------------------------------------------
 Subquery Scan on __unnamed_subquery__
   Filter: ((__unnamed_subquery__.rn >= 5) AND (__unnamed_subquery__.rn <= 10))
   ->  Rownum
         ->  Seq Scan on rewrite_rule_hint_t1
(4 rows)
```

33. WINDOWAGG_PUSHDOWN

    The filter criterion of the window function in the parent query can be pushed down to the subquery. This scenario is also controlled by the GUC parameter **rewrite_rule**. If the hint of this rule is not used and the value of **rewrite_rule** is **disable_windowagg_pushdown**, the filter criterion of the window function in the parent query cannot be pushed down to the subquery. If the hint of this rule is used and the value of **rewrite_rule** is **disable_windowagg_pushdown**, the hint precedes the GUC parameter, and the filter criterion of the window function in the parent query can be pushed down to the subquery. This hint and NO_WINDOWAGG_PUSHDOWN are mutually exclusive.

```
gaussdb=# EXPLAIN (costs off) SELECT * FROM (SELECT /*+WINDOWAGG_PUSHDOWN*/
row_number() over() rid, rewrite_rule_hint_t1.a FROM rewrite_rule_hint_t1) WHERE rid BETWEEN 5
AND 10;
                     QUERY PLAN
------------------------------------------------------------
 Subquery Scan on __unnamed_subquery__
   Filter: (__unnamed_subquery__.rid >= 5)
   ->  WindowAgg
         row_number_filter: (row_number() OVER () <= 10)
         ->  Seq Scan on rewrite_rule_hint_t1
(5 rows)
```

34. NO_WINDOWAGG_PUSHDOWN

    The filter criterion of the window function in the parent query cannot be pushed down to the subquery. This scenario is also controlled by the GUC parameter **rewrite_rule**. If the hint of this rule is used or the value of **rewrite_rule** is **disable_windowagg_pushdown**, the filter criterion of the window function in the parent query cannot be pushed down to the subquery. This hint and WINDOWAGG_PUSHDOWN are mutually exclusive.

```
gaussdb=# EXPLAIN (costs off) SELECT * FROM (SELECT /*+NO_WINDOWAGG_PUSHDOWN*/
row_number() over() rid, rewrite_rule_hint_t1.a FROM rewrite_rule_hint_t1) WHERE rid BETWEEN 5
AND 10;
                              QUERY PLAN
-------------------------------------------------------------------------------
 Subquery Scan on __unnamed_subquery__
   Filter: ((__unnamed_subquery__.rid >= 5) AND (__unnamed_subquery__.rid <= 10))
   ->  WindowAgg
         ->  Seq Scan on rewrite_rule_hint_t1
(4 rows
```

6.9.23 Outline Hints

Description

An outline is an important means of describing an execution plan and a persistent representation of plan fixing. An outline is stored in a database and needs to be compatible between versions. It can guide an optimizer to generate a specified

plan. When generating an execution plan, the kernel can generate an outline. In addition, the optimizer can use the outline to control a plan. The outline is the core prerequisite capability of plan management.

Outline hints are generated by the optimizer to reproduce a plan. They start with **BEGIN_OUTLINE_DATA** and end with **END_OUTLINE_DATA**. You can obtain outline hints by using EXPLAIN(OUTLINE ON). The obtained outline hints can be used to control the plan.

Feature Constraints

1. Ensure that **set explain_perf_mode** is set to **pretty**.
2. An outline is used for planned reproduction and restoration. Currently, the outline can control the following aspects of the same SQL statement:
   - Query rewriting.
   - Physical operators of subquery at each layer:
     (a) Scanning mode
     (b) Joining method
     (c) Joining sequence
     (d) Index table for bitmap scan
     (e) Parameterized path
     (f) Materialization of joined inner tables
   - Aggregation method of subquery at each layer.
   - Additional processing for ANY sublink pullup: hashed or material.
   - Transmission mode of SMP data.
3. Currently, bitmap scan and index scan hints of the kernel specify that the optimizer uses the specified index to generate the index scanning path when scanning related tables. The specific index conditions are generated by the optimizer based on the cost.
4. For complex multi-table join SQL statements, the performance of outline fixed plan restoration is better than that of genetic algorithm.
5. When there are outline hints, for hints that are not between BEGIN OUTLINE and END OUTLINE, if hints are generated by a control plan (hints mentioned in points 2, 3, and 4), they will become invalid. If hints are not generated by the control plan, they are retained, for example, hints of the slow SQL control rule wlmrule.

Syntax

Outline hints comply with the hint syntax.

```
BEGIN_OUTLINE_DATA
VERSION(@version_num)
END_OUTLINE_DATA
```

Parameters

- **@version_num**: specifies an outline version. If not specified, the default value **1.0.0** is used. Currently, only 1.0.0 is supported, which is reserved for outline behavior control in later versions.

- **BEGIN_OUTLINE_DATA** and **END_OUTLINE_DATA**: The generated outline hints must be placed between them.

NOTICE

1. **BEGIN_OUTLINE_DATA** and **END_OUTLINE_DATA** must be used in pairs.
2. Only hints between **BEGIN_OUTLINE_DATA** and **END_OUTLINE_DATA** take effect.

Usage Guide

1. Generate outline hints.

   Setting before use:

   ```
   SET explain_perf_mode = pretty;
   ```

   explain (Outline on):

   ```
   -- Create tables.
   gaussdb=# CREATE TABLE ot_t1(a int, b int);
   gaussdb=# CREATE TABLE ot_t2(a int, b int);

   -- Execute.
   gaussdb=# EXPLAIN (OUTLINE ON, COSTS OFF) SELECT * FROM ot_t1 JOIN ot_t2 ON ot_t1.a =
   ot_t2.a;
    id |          operation
   ----+-------------------------------
     1 | ->  Hash Join (2,3)
     2 |    ->  Seq Scan on ot_t1
     3 |    ->  Hash
     4 |       ->  Seq Scan on ot_t2
   (4 rows)

    Predicate Information (identified by plan id)
   -----------------------------------------------
     1 --Hash Join (2,3)
           Hash Cond: (ot_t1.a = ot_t2.a)
   (2 rows)

                   ====== Outline Data =====
   ---------------------------------------------------------------------
     begin_outline_data
     HashJoin(@"sel$1" public.ot_t1@"sel$1" public.ot_t2@"sel$1")
     Leading(@"sel$1" (public.ot_t1@"sel$1" public.ot_t2@"sel$1"))
     TableScan(@"sel$1" public.ot_t1@"sel$1")
     TableScan(@"sel$1" public.ot_t2@"sel$1")
     version("1.0.0")
     end_outline_data
   (7 rows)
   ```


NOTE

1. The preceding outline data consists of a series of hints. The hints and specified functions of query blocks in hints comply with the original hint capability.
2. **BEGIN_OUTLINE_DATA** and **END_OUTLINE_DATA** indicate the start and end of the outline, respectively.
3. **HashJoin(@"sel$1" t1@"sel$1" t2@"sel$1")** indicates that the hash join operation is performed on **t1** (originally in the **sel$1** query block) and **t2** (originally in the **sel$1** query block) in the first query block (only one query block). **Leading(@"sel$1" (t1@"sel$1" t2@"sel$1"))** indicates the join sequence. **TableScan(@"sel$1" t1@"sel$1")** and **TableScan(@"sel$1" t2@"sel$1")** indicate that sequential scans are performed on both **t1** and **t2**.
4. Outline can correspond to the plan.
5. Currently, the version number is fixed at 1.0.0.

2. Use outline hints.

   When the outline is used, the SQL statements converted from the outline are used.

```
gaussdb=# EXPLAIN (OUTLINE ON, COSTS OFF) SELECT /*+
    BEGIN_OUTLINE_DATA
    HashJoin(@"sel$1" public.ot_t1@"sel$1" public.ot_t2@"sel$1")
    Leading(@"sel$1" (public.ot_t1@"sel$1" public.ot_t2@"sel$1"))
    TableScan(@"sel$1" public.ot_t1@"sel$1")
    TableScan(@"sel$1" public.ot_t2@"sel$1")
    VERSION("1.0.0")
    END_OUTLINE_DATA */ * FROM ot_t1 JOIN ot_t2 ON ot_t1.a = ot_t2.a;
 id |          operation
----+-----------------------------
  1 | ->  Hash Join (2,3)
  2 |    ->  Seq Scan on ot_t1
  3 |    ->  Hash
  4 |       ->  Seq Scan on ot_t2
(4 rows)

 Predicate Information (identified by plan id)
-----------------------------------------------
   1 --Hash Join (2,3)
         Hash Cond: (ot_t1.a = ot_t2.a)
(2 rows)

                  ====== Outline Data =====
-------------------------------------------------------------------
   begin_outline_data
   HashJoin(@"sel$1" public.ot_t1@"sel$1" public.ot_t2@"sel$1")
   Leading(@"sel$1" (public.ot_t1@"sel$1" public.ot_t2@"sel$1"))
   TableScan(@"sel$1" public.ot_t1@"sel$1")
   TableScan(@"sel$1" public.ot_t2@"sel$1")
   version("1.0.0")
   end_outline_data
(7 rows)
```

   Compare 1 and 2. You can see that the two plans are the same, indicating that outline hints can be used to control the generation of plans.

3. Compare plans with and without outline hints.

   a. Common hints:

```
gaussdb=#  EXPLAIN (OUTLINE ON, COSTS OFF) SELECT /*+
    NestLoop(@"sel$1" ot_t1@"sel$1" ot_t2@"sel$1")
    Leading(@"sel$1" (ot_t1@"sel$1" ot_t2@"sel$1"))
    TableScan(@"sel$1" ot_t1@"sel$1")
    TableScan(@"sel$1" ot_t2@"sel$1") */ * FROM ot_t1 JOIN ot_t2 ON ot_t1.a = ot_t2.a;
 id |          operation
----+-----------------------------
  1 | ->  Nested Loop (2,3)
```



```
 2 |    ->  Seq Scan on ot_t1
 3 |    ->  Materialize
 4 |      ->  Seq Scan on ot_t2
(4 rows)

 Predicate Information (identified by plan id)
-----------------------------------------------
  1 --Nested Loop (2,3)
        Join Filter: (ot_t1.a = ot_t2.a)
(2 rows)

                  ====== Outline Data =====
-------------------------------------------------------------------
  begin_outline_data
  NestLoop(@"sel$1" public.ot_t1@"sel$1" public.ot_t2@"sel$1")
  Leading(@"sel$1" (public.ot_t1@"sel$1" public.ot_t2@"sel$1"))
  TableScan(@"sel$1" public.ot_t1@"sel$1")
  Materialize_Inner(@"sel$1" public.ot_t2@"sel$1")
  TableScan(@"sel$1" public.ot_t2@"sel$1")
  version("1.0.0")
  end_outline_data
(8 rows)
```

b. Outline hints:

```
gaussdb=#  EXPLAIN (OUTLINE ON, COSTS OFF) SELECT /*+
    BEGIN_OUTLINE_DATA
    NestLoop(@"sel$1" ot_t1@"sel$1" ot_t2@"sel$1")
    Leading(@"sel$1" (ot_t1@"sel$1" ot_t2@"sel$1"))
    TableScan(@"sel$1" ot_t1@"sel$1")
    TableScan(@"sel$1" ot_t2@"sel$1")
    VERSION("1.0.0")
    END_OUTLINE_DATA */ * from ot_t1 join ot_t2 on ot_t1.a = ot_t2.a;
 id |        operation
----+--------------------------
  1 | ->  Nested Loop (2,3)
  2 |    ->  Seq Scan on ot_t1
  3 |    ->  Seq Scan on ot_t2
(3 rows)

 Predicate Information (identified by plan id)
-----------------------------------------------
  1 --Nested Loop (2,3)
        Join Filter: (ot_t1.a = ot_t2.a)
(2 rows)

                  ====== Outline Data =====
-------------------------------------------------------------------
  begin_outline_data
  NestLoop(@"sel$1" public.ot_t1@"sel$1" public.ot_t2@"sel$1")
  Leading(@"sel$1" (public.ot_t1@"sel$1" public.ot_t2@"sel$1"))
  TableScan(@"sel$1" public.ot_t1@"sel$1")
  TableScan(@"sel$1" public.ot_t2@"sel$1")
  version("1.0.0")
  end_outline_data
(7 rows)
```

NOTE

As you can see, common hints specify only part of the behavior and cannot completely fix the plan.

In example **a**, common hints generate a materialize plan, which is not in the hints.

In example **b**, outline hints do not generate the materialize plan and completely fix the plan.

6.9.24 Hints for Specifying ANY Sublink Pullup

Description

A method of optimizing operators is specified when an ANY sublink is pulled up.

Syntax

```
[no] hashed_sublink[(@queryblock)], material_sublink[(@queryblock)]
```

Parameters

- **no** indicates that the optimization is not used.
- For details about **@queryblock**, see **Hint for Specifying the Query Block Where the Hint Is Located**. This parameter can be omitted, indicating that it takes effect in the current query block. If it is not specified, the hint does not have parentheses (()).
- **hashed_sublink** uses hash table optimization for specified sublinks.
- **material_sublink** uses materialized optimization for specified sublinks.

Examples

- material_sublink

  Table creation statements:

```
CREATE TABLE SUPPLIER (
    S_SUPPKEY BIGINT NOT NULL,
    S_NAME CHAR(25) NOT NULL,
    S_ADDRESS VARCHAR(40) NOT NULL,
    S_NATIONKEY INT NOT NULL,
    S_PHONE CHAR(15) NOT NULL,
    S_ACCTBAL DECIMAL(15, 2) NOT NULL,
    S_COMMENT VARCHAR(101) NOT NULL,
    S_STATEKEY INT NOT NULL
);
set explain_perf_mode=pretty; -- Open the explain pretty option to view a detailed plan.
```

  Example:

```
-- material_sublink hint
gaussdb=#  EXPLAIN (blockname on, costs off) SELECT max(s_acctbal) nation_total, s_nationkey
FROM supplier WHERE s_suppkey < 10000
GROUP BY s_nationkey having nation_total = ANY (SELECT /*+ material_sublink */ avg(s_acctbal)
FROM supplier GROUP BY s_statekey);
 id |                operation                 | Query Block
----+------------------------------------------+-------------
  1 | ->  HashAggregate                        | sel$1
  2 |    ->  Seq Scan on supplier@"sel$1"      | sel$1
  3 |    ->  Materialize  [1, SubPlan 1]       |
  4 |       ->  HashAggregate                  | sel$2
  5 |          ->  Seq Scan on supplier@"sel$2" | sel$2
(5 rows)

 Predicate Information (identified by plan id)
-----------------------------------------------
  1 --HashAggregate
        Filter: (SubPlan 1)
  2 --Seq Scan on supplier@"sel$1"
        Filter: (s_suppkey < 10000)
(4 rows)
```

  The material operation is performed on the sublink. If the hint is not used, hashed is used.

- hashed_sublink

  Table creation and preparation:

```
set work_mem='64kB';  -- Reduce the work memory to reproduce the scenario.
set explain_perf_mode = pretty;   -- Open the explain pretty option to view a detailed plan.
CREATE TABLE nt1 (a int);
CREATE TABLE nt2 (a int);
INSERT INTO nt1 VALUES(generate_series(1, 50000));
INSERT INTO nt2 VALUES(generate_series(1, 50000));
```

  Example:

```
-- hashed_sublink hint
gaussdb=# EXPLAIN SELECT * FROM nt1 WHERE nt1.a NOT IN (SELECT /*+ hashed_sublink
no_expand*/ a FROM nt2);
 id |               operation                | E-rows | E-width |      E-costs
----+----------------------------------------+--------+---------+--------------------
  1 | ->  Seq Scan on nt1                    |  49869 |       4 | 1627.725..3255.450
  2 |    ->  Seq Scan on nt2  [1, SubPlan 1] |  99738 |       4 | 0.000..1378.380
(2 rows)

 Predicate Information (identified by plan id)
-----------------------------------------------
   1 --Seq Scan on nt1
         Filter: (NOT (hashed SubPlan 1))
(2 rows)
-- work_mem affects plan generation and execution. If you perform other operations, you need to
reset it.
reset work_mem;  -- Roll back the memory settings.
```

  The hashed operation is performed on the sublink. If the hint is not used, material is used.

6.9.25 Hints for Specifying the Degree of Parallelism for Scans

Description

The degree of parallelism (DOP) is specified for table scans in a parallel execution plan.

Syntax

```
scandop([@queryblock] table dop_num)
```

Parameters

- For details about **@queryblock**, see **Hint for Specifying the Query Block Where the Hint Is Located**. This parameter can be omitted, indicating that it takes effect in the current query block.
- **table** specifies the table to be scanned. You can specify only one table. Use a table alias (if any) instead of a table name.
- **dop_num** specifies the DOP for table scans.
- **scandop** specifies a hint for specifying the DOP for scans.

Examples

```
-- Preparation
CREATE TABLE cst1(a int, b int, c int, d bigint);
set explain_perf_mode = pretty;  -- Open the explain pretty option to view a detailed plan.
-- Usage
gaussdb=# EXPLAIN (costs off) SELECT /*+ Set(query_dop 2) scandop(cst1 2)*/ * FROM cst1;
 id |              operation
----+-----------------------------------------
```



```
 1 | ->  Streaming(type: LOCAL GATHER dop: 1/2)
 2 |    ->  Seq Scan on cst1
(2 rows)
```

In a parallel plan, you can use scandop hints to specify the degree of parallelism.

NOTE

The hint takes effect only when **dop_num** is the same as the current degree of parallelism (**query_dop**) or is set to **1**.

```
gaussdb=# EXPLAIN (costs off) SELECT /*+ Set(query_dop 2) scandop(cst1 4)*/ * FROM cst1;
 id |      operation
----+----------------------
 1 | ->  Seq Scan on cst1
(1 row)
```

In some scenarios with subqueries, the Stream operator deteriorates and is eliminated. In this case, although the scandop hint takes effect, no stream plan is generated.

```
gaussdb=# EXPLAIN(costs off) SELECT /*+ Set(query_dop 2) scandop(cst1 2) */ count(a) FROM cst1
WHERE b > (SELECT max(a) FROM cst1 ORDER BY c LIMIT 1);
 id |                 operation
----+-----------------------------------------------
 1 | ->  Aggregate
 2 |    ->  Aggregate
 3 |       ->  Seq Scan on cst1
 4 |    ->  Limit  [3, InitPlan 1 (returns $0)]
 5 |       ->  Aggregate
 6 |          ->  Seq Scan on cst1
(6 rows)

 Predicate Information (identified by plan id)
------------------------------------------------
   3 --Seq Scan on cst1
        Filter: (b > $0)
(2 rows)
```

Compare it with a plan without hints.

```
 id |                 operation
----+-----------------------------------------------
 1 | ->  Aggregate
 2 |    ->  Seq Scan on cst1
 3 |    ->  Limit  [2, InitPlan 1 (returns $0)]
 4 |       ->  Aggregate
 5 |          ->  Seq Scan on cst1
(5 rows)

 Predicate Information (identified by plan id)
------------------------------------------------
   2 --Seq Scan on cst1
        Filter: (b > $0)
(2 rows)
```

You can see that the plan with scandop hint has an additional Aggregate operator, which is the residual after the Stream operator is eliminated.

A common plan has only one layer of Aggregate operator. The two plans do not affect the final result.

6.9.26 Hints for Specifying Whether to Use Semi-Join

Description

Specifies whether to select semi-join.

Syntax

```
[no] semijoin([@queryblock] table_list)
```


Parameters

- **no** indicates that semi-join is not used.
- For details about **@queryblock**, see **Hint for Specifying the Query Block Where the Hint Is Located**. This parameter can be omitted, indicating that it takes effect in the current query block.
- **table_list** specifies the tables to be joined. The values are the same as those of **join_table_list** but contain no parentheses.
- **semijoin** specifies whether semi-join is used for table join.

Examples

```
-- Preparation
CREATE TABLE se_t1 (a int, b int);
CREATE TABLE se_t2 (a int, b int);
CREATE TABLE se_t3 (a int, b int);
set explain_perf_mode = pretty;  -- Open the explain pretty option to view a detailed plan.
-- Scenarios where hints are not used
gaussdb=# EXPLAIN(costs off) SELECT a FROM se_t2 WHERE a IN( SELECT  se_t1.a FROM se_t1,se_t3
WHERE se_t1.a=se_t3.a GROUP BY 1);
 id |             operation
----+---------------------------------------
  1 | ->  Hash Semi Join (2, 3)
  2 |    ->  Seq Scan on se_t2
  3 |    ->  Hash
  4 |       ->  HashAggregate
  5 |          ->  Hash Join (6,7)
  6 |             ->  Seq Scan on se_t1
  7 |             ->  Hash
  8 |                ->  Seq Scan on se_t3
(8 rows)

 Predicate Information (identified by plan id)
------------------------------------------------
   1 --Hash Semi Join (2, 3)
         Hash Cond: (se_t2.a = se_t1.a)
   5 --Hash Join (6,7)
         Hash Cond: (se_t1.a = se_t3.a)
(4 rows)

-- Use the semijoin hint to disable semi-join.
gaussdb=# EXPLAIN(costs off) SELECT /*+ NO SemiJoin(se_t2@"sel$1" "sel$2") */ a FROM se_t2 WHERE a
IN( SELECT  se_t1.a FROM se_t1,se_t3 WHERE se_t1.a=se_t3.a GROUP BY 1);
 id |             operation
----+---------------------------------------
  1 | ->  Hash Join (2,3)
  2 |    ->  Seq Scan on se_t2
  3 |    ->  Hash
  4 |       ->  HashAggregate
  5 |          ->  Hash Join (6,7)
  6 |             ->  Seq Scan on se_t1
  7 |             ->  Hash
  8 |                ->  Seq Scan on se_t3
(8 rows)

 Predicate Information (identified by plan id)
------------------------------------------------
   1 --Hash Join (2,3)
         Hash Cond: (se_t2.a = se_t1.a)
   5 --Hash Join (6,7)
         Hash Cond: (se_t1.a = se_t3.a)
(4 rows)
```

You can see that the semijoin hint can determine whether to use semi-join.

6.9.27 Hints for Specifying the Stream Hashagg Optimization

Description

In a parallel execution plan, if the column used by an operator is different from that used by GROUP BY and the hash operator is used, redistribution is optimized for the plan. In this case, you can use this hint to control the generation of a plan.

Syntax

```
redistribute_agg[(@queryblock)], agg_redistribute_agg[(@queryblock)]
```

Parameters

- For details about **@queryblock**, see **Hint for Specifying the Query Block Where the Hint Is Located**. This parameter can be omitted, indicating that it takes effect in the current query block.
- **redistribute_agg** specifies that stream hashagg uses the redistribution-aggregation plan.
- **agg_redistribute_agg** specifies that stream hashagg uses the aggregation-redistribution-aggregation plan.

Examples

```
-- Preparation
CREATE TABLE agg_t1 (a int, b int, c int);
set explain_perf_mode=pretty;  -- Open the explain pretty option to view a detailed plan.
set query_dop = 2; -- Set the degree of parallelism.

-- Do not use stream agg hints.
gaussdb=# EXPLAIN (costs off) SELECT /*+ scandop(agg_t1 2)*/ a, avg(b), sum(c)
   FROM agg_t1
   GROUP by a
   ORDER by 1,2,3;
 id |                       operation
----+-------------------------------------------------------
  1 | ->  Sort
  2 |    ->  Streaming(type: LOCAL GATHER dop: 1/2)
  3 |       ->  HashAggregate
  4 |          ->  Streaming(type: LOCAL REDISTRIBUTE dop: 2/2)
  5 |             ->  HashAggregate
  6 |                ->  Seq Scan on agg_t1
(6 rows)

-- Use redistribute_agg hints.
gaussdb=# EXPLAIN (costs off) SELECT /*+ redistribute_agg scandop(agg_t1 2)*/ a, avg(b), sum(c)
   FROM agg_t1
   GROUP BY a
   ORDER BY 1,2,3;
 id |                       operation
----+-------------------------------------------------------
  1 | ->  Sort
  2 |    ->  Streaming(type: LOCAL GATHER dop: 1/2)
  3 |       ->  HashAggregate
  4 |          ->  Streaming(type: LOCAL REDISTRIBUTE dop: 2/2)
  5 |             ->  Seq Scan on agg_t1
(5 rows)
```

You can see that redistribute_agg hints can change the plan to redistribution-aggregation instead of aggregation-redistribution-aggregation.

6.9.28 Hints for Specifying Whether to Use Minmax Optimization

Description

Specifies whether a statement is rewritten using minmax.

Syntax

```
[no] use_minmax[(@queryblock)]
```

Parameters

- **no** indicates that minmax is not used for query rewriting.
- For details about **@queryblock**, see **Hint for Specifying the Query Block Where the Hint Is Located**. This parameter can be omitted, indicating that it takes effect in the current query block.
- **use_minmax** uses minmax optimization for statement query rewriting.

Examples

```
-- Preparation
create table minmaxtest(f1 int);
create index minmaxtesti on minmaxtest(f1);
insert into minmaxtest values(11), (12);
set explain_perf_mode=pretty;  -- Open the explain pretty option to view a detailed plan.

-- Common scenario
gaussdb=# explain (costs off) select min(f1), max(f1) from minmaxtest;
 id |                          operation
----+------------------------------------------------------------------------
  1 | ->  Result
  2 |    ->  Limit  [1, InitPlan 1 (returns $0)]
  3 |       ->  Index Only Scan using minmaxtesti on minmaxtest
  4 |    ->  Limit  [1, InitPlan 2 (returns $1)]
  5 |       ->  Index Only Scan Backward using minmaxtesti on minmaxtest
(5 rows)

        Predicate Information (identified by plan id)
-------------------------------------------------------------------
   3 --Index Only Scan using minmaxtesti on minmaxtest
        Index Cond: (f1 IS NOT NULL)
   5 --Index Only Scan Backward using minmaxtesti on minmaxtest
        Index Cond: (f1 IS NOT NULL)
(4 rows)

-- Use hints to disable minmax rewriting.
gaussdb=# explain (costs off)select /*+ no use_minmax*/ min(f1), max(f1) from minmaxtest;
 id |          operation
----+-----------------------------
  1 | ->  Aggregate
  2 |    ->  Seq Scan on minmaxtest
(2 rows)
```

After the no use_minmax hint is used, the SQL statement does not use minmax optimization.

```
-- Use minmax hints.
analyze;  -- Collect statistics.
gaussdb=# explain (costs off) select min(f1), max(f1) from minmaxtest;

 id |          operation
```



```
----+------------------------------
  1 | ->  Aggregate
  2 |    ->  Seq Scan on minmaxtest
(2 rows)

-- Use the use_minmax hint to select minmax rewriting.
gaussdb=# explain (costs off) select  /*+ indexonlyscan(minmaxtest) use_minmax*/ min(f1), max(f1) from
minmaxtest;
 id |                          operation
----+-------------------------------------------------------------------------
  1 | ->  Result
  2 |    ->  Limit  [1, InitPlan 1 (returns $0)]
  3 |       ->  Index Only Scan using minmaxtesti on minmaxtest
  4 |    ->  Limit  [1, InitPlan 2 (returns $1)]
  5 |       ->  Index Only Scan Backward using minmaxtesti on minmaxtest
(5 rows)

        Predicate Information (identified by plan id)
-----------------------------------------------------------------------
   3 --Index Only Scan using minmaxtesti on minmaxtest
         Index Cond: (f1 IS NOT NULL)
   5 --Index Only Scan Backward using minmaxtesti on minmaxtest
         Index Cond: (f1 IS NOT NULL)
(4 rows)
```

The use_minmax hint takes effect.

NOTE

The use_minmax optimization takes effect only when index scan is used for table scanning.

6.10 Introduction to Plan Trace

NOTICE

1. This feature is used by database kernel developers for in-depth analysis of slow SQL statements. It is not recommended that non-kernel developers use this feature.
2. After this feature is enabled, optimizer information is recorded in the system catalog during DML execution. As a result, the read transaction becomes a write transaction, and functions that must be executed in the read transaction, such as pg_create_logical_replication_slot, cannot be executed.

You can use the plan trace feature to view the optimization process of a query plan. In the plan trace, you can view key information such as the path calculation process, path selection process, and path elimination process in the plan to analyze the root cause of slow SQL statements. The following describes the two methods of using this feature:

```
-- Prepare tables.
CREATE TABLE tb_a(c1 int);
CREATE TABLE tb_b(c1 int);
CREATE INDEX tb_a_idx_c1 ON tb_a(c1);
CREATE INDEX idx_b ON tb_b(c1);
```

Method 1: Use the GUC parameter **enable_plan_trace** to enable the plan trace feature. The procedure is as follows:

Step 1 Enable the plan trace GUC function.


```
SET enable_plan_trace = on;
```

Step 2 Run the service SQL statement. For example, the service SQL statement is as follows:

```
SELECT * FROM tb_a a, tb_b b WHERE a.c1 = b.c1 AND a.c1=1;
```

Step 3 View the newly generated plan trace in the gs_my_plan_trace view.

```
SELECT * FROM gs_my_plan_trace ORDER BY modifydate LIMIT 1;
```

Generally, plan trace records are large. If gsql is used to connect to the database, you are advised to run the **\x** command to change the display mode of gsql query results to **Expanded**.

The plan trace records are usually large. Therefore, only fragments of the key trace execution result of this example are provided.

Fragment 1: In the trace, you can view the SQL statement and plan that are being executed.

```
query_id      | 4f078a966a1c4a434167b2f780bbfd92
query         | select * from tb_a a, tb_b b where a.c1 = b.c1 and a.c1=1;
unique_sql_id | 2108646922
plan          | Datanode Name: datanode
              | Nested Loop  (cost=0.00..81.88 rows=144 width=8)
              |   ->  Seq Scan on tb_a a  (cost=0.00..40.03 rows=12 width=4)
              |         Filter: (c1 = '***')
              |   ->  Materialize  (cost=0.00..40.09 rows=12 width=4)
              |         ->  Seq Scan on tb_b b  (cost=0.00..40.03 rows=12 width=4)
              |               Filter: (c1 = '***')
```

Fragment 2: In the trace, you can view the key GUC parameters used by the current SQL statement.

```
plan_trace    | [key_guc]
              | enable_pbe_optimization=1
              | plan_cache_mode=0
              | random_page_cost=4.000
              | enable_hashjoin=1
              | enable_mergejoin=1
              | enable_nestloop=1
              | enable_seqscan=1
              | effective_cache_size=16385
              | work_mem=65536
              | default_statistics_target=100
              | cost_param=0
              | =[key_guc]=
```

Fragment 3: In the trace, you can view the process of calculating the path cost of the current SQL query plan.

```
              | [optcost_initial_cost_nestloop]
              | method_initial_state: inner_pathid,2 outer_pathid,1 inner_start_cost,0.000000
inner_total_cost,40.025000 outer_start_cost,0.000000 outer_total_cost,40.025000 outer_path_rows,12.000000
              | cal: inner_rescan_start_cost,0.000000 inner_rescan_total_cost,40.025000
              | cal: inner_run_cost = inner_total_cost - inner_start_cost 40.025000, 40.025000, 0.000000
              | cal: inner_rescan_run_cost = inner_rescan_total_cost - inner_rescan_start_cost 40.025000
              | cal: startup_cost += outer_start_cost + inner_start_cost 0.000000
              | cal: run_cost += outer_total_cost - outer_start_cost 40.025000
              | cal: run_cost += (outer_path_rows - 1) * inner_rescan_start_cost 40.025000
              | cal: run_cost += inner_run_cost 80.050000
              | cal: run_cost += (outer_path_rows - 1) * inner_rescan_run_cost 520.325000
              | Initial nestloop cost: startup_cost: 0.000000, total_cost: 520.325000
              | =[optcost_initial_cost_nestloop]=
```

Fragment 4: In the trace, you can see the elimination process of the base table path: 1. The old path is eliminated. 2. Reasons why the old path was eliminated are checked; 3. Information about the new path is checked.


```
            | An old path is removed with cost = 0.000000 .. 521.765000;  rows = 144.000000
            | The old path and the comparison results are:
            | {
            |         old pathid=00000004     Cost = NewBetter        |       PathKeys = Equal        |          BMS =
Equal          |         Rows = Equal
            | }
            | A new path is accepted with cost = 0.000000 .. 81.880000;  rows = 144.000000
            | The detail information of the new path:
            | {
            |         NestLoop(1:tb_a  2:tb_b ) pathid=00000005 hasparam=0 rows=144 multiple=1.000000
tuples=0.00 rpages=0.00 ipages=0.00 selec=0.00000000 ml=0 iscost=1 lossy=0 uidx=0)  dop=1
cost=0.00..81.88 hint 0 trace_id=#1##3##5#       clauses:  outerpathid=00000001 innerpathid=00000003
            | }
```

----End

Method 2: Use the system function gs_plan_trace_watch_sqlid to enable the plan trace feature. The procedure is as follows:

Step 1 Obtain the unique SQL ID of the SQL statement from the dbe_perf.statement system catalog. For example, run the following SQL statement to obtain the unique SQL ID:

```
SELECT * FROM dbe_perf.statement WHERE query LIKE '%tb_a%';
```

The value of **unique_sql_id** in the command output is as follows:

```
node_name            | datanode1
node_id              | 0
user_name            | qiumc
user_id              | 10
unique_sql_id        | 1921680825
query                | select * from tb_a a, tb_b b where a.id=b.id and a.c1=?;
n_calls              | 3
min_elapse_time      | 8880
max_elapse_time      | 12371
total_elapse_time    | 32036
```

Step 2 A user with the sysadmin permission calls the gs_plan_trace_watch_sqlid function to listen to the unique SQL ID. For example:

```
SELECT gs_plan_trace_watch_sqlid(1921680825);
```

Step 3 If no plan trace is generated for the unique SQL ID, the unique SQL ID is saved in a memory list. You can use the gs_plan_trace_show_sqlids() function to view the unique SQL ID list of the plan trace to be collected. An example SQL statement is as follows:

```
SELECT gs_plan_trace_show_sqlids();
```

The execution result of the SQL statement is as follows:

```
-[ RECORD 1 ]--------------+------------
gs_plan_trace_show_sqlids | 1921680825,
```

Step 4 If you run the following SQL statement:

```
SELECT * FROM tb_a a, tb_b b WHERE a.c1=b.c1 AND a.c1=1;
```

You can also generate plan trace records for the SQL statement. The subsequent steps are the same as those in step 3 in method 1.

NOTICE

Only users with the sysadmin, opradmin, or monadmin permission can call the gs_plan_trace_watch_sqlid and gs_plan_trace_show_sqlids functions. If a common user executes the SQL statement with the unique SQL ID listened by the administrator, the common user can use the gs_my_plan_trace view to view the plan trace generated by the common user.

----End

NOTICE

Generally, plan trace records are large. You need to clear them in a timely manner. Otherwise, a large amount of disk space is occupied. You can use the gs_plan_trace_delete function to delete the plan trace records generated by yourself.

For example, run the following SQL statement:

```
select gs_plan_trace_delete(TIMESTAMPTZ '2023-01-10 17:16:42.652543+08')
```

You can delete all plan trace records earlier than or equal to the 2023-01-10 17:16:42.652543+08 for the current user. In this way, each user can delete its own plan trace data.

6.11 Tuning with SQL Patches

SQL patches are designed for database administrators (DBAs), O&M personnel, and other roles who need to optimize SQL statements. If performance problems caused by poor plans of service statements are identified through other O&M views or fault locating methods, you can create SQL patches to optimize service statements based on hints. Currently, the following hints are supported: number of rows, scanning mode, join mode, join sequence, PBE custom/generic plan selection, statement-level parameter setting, and parameterized path. In addition, in case that services are unavailable due to internal system errors that are triggered by specific statements, you can create SQL patches to rectify single-point failures without changing service statements. In this way, errors can be reported in advance to avoid greater loss.

Feature Constraints

1. Patches can be created only by unique SQL ID. If unique SQL IDs conflict, SQL patches that are used for hint-based optimization may affect performance but do not affect semantic correctness.
2. Only hints that do not change SQL semantics can be used as patches. SQL rewriting is not supported.
3. This tool is not applicable to logical backup and restoration.
4. The patch validity cannot be verified during patch creation. If the patch hint has syntax or semantic errors, the query execution is not affected.
5. Only the initial user, O&M administrator, monitor administrator, and system administrator have the permission to perform this operation.

6. Patches are not shared between databases. When creating SQL patches, you need to connect to the target database.
7. In the centralized deployment scenario where the standby node is readable, you must specify the primary node to call functions to create, modify, or delete SQL patches and the standby node to report errors.
8. There is a delay in synchronizing an SQL patch to the standby node. The patch takes effect after the standby node replays related logs.
9. SQL patches in a stored procedure and global SQL patches cannot coexist.
10. SQL patches cannot be used for precompiled statements that are executed using the PREPARE + EXECUTE syntax. For details about special cases, see **Special Cases**.
11. It is not recommended that the SQL patches be used in the database for a long time. It should be used only as a workaround. If the database service is unavailable due to a kernel fault triggered by a specific statement or SQL hints are used for performance tuning, you must rectify the service fault or upgrade the kernel as soon as possible. After the upgrade, the method of generating unique SQL IDs may change. Therefore, the workaround may become invalid.
12. Currently, except DML statements, unique SQL IDs of SQL statements (such as CREATE TABLE) are generated by hashing the statement text. Therefore, SQL patches are sensitive to uppercase and lowercase letters, spaces, and linefeeds. That is, even statements of different texts have the same semantics, you still need to create different SQL patches for them. For DML operations, an SQL patch can take effect for the same statement with different input parameters, regardless of uppercase letters, lowercase letters, and spaces.

Example

The SQL patch is implemented based on the unique SQL ID. Therefore, you need to enable related O&M parameters (**enable_resource_track = on**, **instr_unique_sql_count > 0**) for the SQL patch to take effect. The unique SQL ID can be obtained from both the WDR and slow SQL view. You need to specify the unique SQL ID when creating the SQL patch. For SQL statements in a stored procedure, you need to set **instr_unique_sql_track_type** to **'all'** and query unique SQL ID in the dbe_perf.statement_history view.

The following provides simple examples:

Scenario 1: Use an SQL patch to optimize specific statements based on hints.

```
gaussdb=# CREATE TABLE hint_t1(a int, b int, c int);
CREATE TABLE
gaussdb=# CREATE INDEX ON hint_t1(a);
CREATE INDEX
gaussdb=# INSERT INTO hint_t1 VALUES(1,1,1);
INSERT 0 1
gaussdb=# sEt track_stmt_stat_level = 'L1,L1'; -- Enable full SQL statistics.
SET
gaussdb=# SET explain_perf_mode = normal;
SET
gaussdb=# SELECT * FROM hint_t1 t1 WHERE t1.a = 1; -- Execute the SQL statement.
 a | b | c
---+---+---
 1 | 1 | 1
(1 row)
gaussdb=# \x -- Switch to the extended display mode to facilitate plan observation.
```



```
Expanded display is on.
gaussdb=# SELECT unique_query_id, query, query_plan FROM dbe_perf.statement_history WHERE query
LIKE '%hint_t1%'; -- Obtain the query plan and unique SQL ID.
-[ RECORD 1 ]---+-----------------------------------------------------------------------------
unique_query_id | 2311517824
query           | select * from hint_t1 t1 where t1.a = ?;
query_plan      | Datanode Name: sgnode
                | Bitmap Heap Scan on hint_t1 t1  (cost=4.33..14.88 rows=10 width=12)
                |   Recheck Cond: (a = '***')
                |   ->  Bitmap Index Scan on hint_t1_a_idx  (cost=0.00..4.33 rows=10 width=0)
                |         Index Cond: (a = '***')
                |
                |
gaussdb=# \x
Expanded display is off.
gaussdb=# SELECT * FROM dbe_sql_util.create_hint_sql_patch('patch1', 2311517824, 'indexscan(t1)'); --
Specify a hint patch for the specified unique SQL ID.
 create_hint_sql_patch
-----------------------
 t
(1 row)

gaussdb=# EXPLAIN SELECT * FROM hint_t1 t1 WHERE t1.a = 1; -- Check whether the hint takes effect.
NOTICE:  Plan influenced by SQL hint patch
                                QUERY PLAN
-------------------------------------------------------------------------------
 [Bypass]
 Index Scan using hint_t1_a_idx on hint_t1 t1  (cost=0.00..32.43 rows=10 width=12)
   Index Cond: (a = 1)
(3 rows)
gaussdb=# SELECT * FROM hint_t1 t1 WHERE t1.a = 1; -- Execute the statement again.
 a | b | c
---+---+---
 1 | 1 | 1
(1 row)
gaussdb=# \x
Expanded display is on.
gaussdb=# SELECT unique_query_id, query, query_plan from dbe_perf.statement_history WHERE query LIKE
'%hint_t1%'; -- The query plan has been changed.
-[ RECORD 1 ]---+-----------------------------------------------------------------------------------------------------
unique_query_id | 2311517824
query           | select * from hint_t1 t1 where t1.a = ?;
query_plan      | Datanode Name: sgnode
                | Bitmap Heap Scan on hint_t1 t1  (cost=4.33..15.70 rows=10 p-time=0 p-rows=0 width=12)
                |   Recheck Cond: (a = '***')
                |   ->  Bitmap Index Scan on hint_t1_a_idx  (cost=0.00..4.33 rows=10 p-time=0 p-rows=0 width=0)
                |         Index Cond: (a = '***')
                |
                |
-[ RECORD 2 ]---+-----------------------------------------------------------------------------------------------------
unique_query_id | 2311517824
query           | select * from hint_t1 t1 where t1.a = ?;
query_plan      | Datanode Name: sgnode
                | Index Scan using hint_t1_a_idx on hint_t1 t1  (cost=0.00..8.27 rows=1 p-time=0 p-rows=0
width=12)
                |   Index Cond: (a = '***')
                |
                |
```

Scenario 2: Use an SQL patch to report an error for a specific statement in advance.

```
gaussdb=# SELECT * FROM dbe_sql_util.drop_sql_patch('patch1'); -- Delete patch 1.
 drop_sql_patch
----------------
 t
(1 row)
gaussdb=# SELECT * FROM dbe_sql_util.create_abort_sql_patch('patch2', 2311517824); -- Create an abort
patch for the statement of the unique SQL ID.
 create_abort_sql_patch
```



```
------------------------
 t
(1 row)

gaussdb=# SELECT * FROM hint_t1 t1 WHERE t1.a = 1; -- An error is reported in advance when the
statement is executed again.
ERROR:  Statement 2578396627 canceled by abort patch patch2
```

Scenario 3: Create an SQL patch for SQL statements in a stored procedure.

```
gaussdb=# CREATE TABLE test_proc_patch(a int,b int);
CREATE TABLE
gaussdb=# INSERT INTO test_proc_patch VALUES(1,2);
INSERT 0 1
gaussdb=# CREATE INDEX test_a ON test_proc_patch(a);
CREATE INDEX
gaussdb=# CREATE PROCEDURE mypro() AS num int;
gaussdb$# BEGIN
gaussdb$# SELECT b INTO num FROM test_proc_patch WHERE a = 1;
gaussdb$# END;
gaussdb$# /
CREATE PROCEDURE
gaussdb=# SET track_stmt_stat_level = 'L0,L1';  -- Open the statistics information.
SET
gaussdb=# SELECT b FROM test_proc_patch WHERE a = 1;
 b
---
 2
(1 row)
gaussdb=# CALL mypro();
 mypro
-------

(1 row)
gaussdb=# SET track_stmt_stat_level = 'OFF,L0';    -- Temporarily disable the function of recording statistics.
SET
gaussdb=# SELECT unique_query_id, query, query_plan, parent_unique_sql_id FROM
dbe_perf.statement_history WHERE query LIKE '%call mypro();%' OR query LIKE '%test_proc_patch%';
 unique_query_id |                 query                  |                        query_plan                         |
parent_unique_sql_id
-----------------+----------------------------------------
+-----------------------------------------------------------------+----------------------
     2859505004 | select b from test_proc_patch where a = ?; | Datanode Name:
sgnode                                +|                  0
              |                                        | Seq Scan on test_proc_patch  (cost=0.00..1.01 rows=1 width=4)+|
              |                                        |   Filter: (a = '***')                                          +|
              |                                        |                                                               +|
              |                                        |                                                                |
     3460545602 | call mypro();                             | Datanode Name: sgnode
+|                 0
              |                                        | Function Scan on mypro  (cost=0.25..0.26 rows=1 width=4)     +|
              |                                        |                                                               +|
              |                                        |                                                                |
     2859505004 | select b from test_proc_patch where a = ?; | Datanode Name:
sgnode                                +|          3460545602
              |                                        | Seq Scan on test_proc_patch  (cost=0.00..1.01 rows=1 width=4)+|
              |                                        |   Filter: (a = '***')                                          +|
              |                                        |                                                               +|
              |                                        |                                                                |
(3 rows)

-- The overloaded function can be called based on parentid to restrict the effective range of the SQL patch
inside the stored procedure.
gaussdb=# SELECT * FROM
dbe_sql_util.create_hint_sql_patch('patch1',2859505004,3460545602,'indexscan(test_proc_patch)');
 create_hint_sql_patch
-----------------------
 t
(1 row)
```



```
gaussdb=# SELECT patch_name,unique_sql_id,parent_unique_sql_id,enable,abort,hint_string FROM
gs_sql_patch WHERE patch_name = 'patch1';  -- Verify that the SQL patch record is correct.
 patch_name | unique_sql_id |  parent_unique_sql_id | enable | abort |      hint_string
------------+---------------+----------------------+--------+-------+--------------------------
 patch1     |    2859505004 |           3460545602 | t      | f     | indexscan(test_proc_patch)
(1 row)
gaussdb=# SET track_stmt_stat_level = 'L0,L1';  -- Open the statistics information.
gaussdb=# SELECT b FROM test_proc_patch where a = 1;
 b
---
 2
(1 row)
gaussdb=# call mypro();
 mypro
-------

(1 row)
gaussdb=# SELECT unique_query_id, query, query_plan, parent_unique_sql_id FROM
dbe_perf.statement_history where query like '%test_proc_patch%' order by start_time;
 unique_query_id |                  query                   |                              query_plan
| parent_unique_sql_id
-----------------+------------------------------------------
+---------------------------------------------------------------------------+----------------------
      2859505004 | select b from test_proc_patch where a = ?; | Datanode Name:
sgnode                                                    +|                    0
                 |                                          | Seq Scan on test_proc_patch  (cost=0.00..1.01 rows=1
width=4)             +|
                 |                                          |   Filter: (a = '***')                                     +|
                 |                                          |                                                           +|
                 |                                          |                                                            |
      2859505004 | select b from test_proc_patch where a = ?; | Datanode Name:
sgnode                                                    +|           3460545602
                 |                                          | Seq Scan on test_proc_patch  (cost=0.00..1.01 rows=1
width=4)             +|
                 |                                          |   Filter: (a = '***')                                     +|
                 |                                          |                                                           +|
                 |                                          |                                                            |
      2859505004 | select b from test_proc_patch where a = ?; | Datanode Name:
sgnode                                                    +|                    0
                 |                                          | Seq Scan on test_proc_patch  (cost=0.00..1.01 rows=1
width=4)             +|
                 |                                          |   Filter: (a = '***')                                     +|
                 |                                          |                                                           +|
                 |                                          |                                                            |
      2859505004 | select b from test_proc_patch where a = ?; | Datanode Name:
sgnode                                                    +|           3460545602
                 |                                          | Index Scan using test_a on test_proc_patch  (cost=0.00..8.27
rows=1 width=4)+|
                 |                                          |   Index Cond: (a = '***')                                    +|
                 |                                          |                                                           +|
                 |                                          |                                                            |
(4 rows)
```

Special Cases

According to the example, SQL patches can be used only when the correct unique SQL IDs are used. Therefore, SQL patches do not support precompiled statements executed by the PREPARE + EXECUTE syntax.

```
-- Generally, an SQL ID with PREPARE is obtained. Therefore, the SQL patch cannot be used.
unique_query_id |                         query
-----------------+----------------------------------------------------------
       658407023 | prepare p1 as                                          +
                 | SELECT /*+ tablescan(rewrite_rule_hint_t1)*/*          +
                 | FROM rewrite_rule_hint_t1,                             +
                 |      (SELECT * FROM rewrite_rule_hint_t2 WHERE a > 1) tt+
                 | WHERE rewrite_rule_hint_t1.a = tt.a;
```

However, if the plan cache becomes invalid, the plan cache uses the statement in PREPARE to generate a unique SQL ID again. If the unique SQL ID is used to apply the SQL patch, the SQL patch can be applied normally.

```
-- Example
-- Create a table.
gaussdb=# DROP TABLE rewrite_rule_hint_t1;
gaussdb=# DROP TABLE rewrite_rule_hint_t2;
gaussdb=# CREATE TABLE rewrite_rule_hint_t1 (a int, b int, c int, d int);
gaussdb=# CREATE TABLE rewrite_rule_hint_t2 (a int, b int, c int, d int);

-- Enable FullSQL statistics.
gaussdb=# SET track_stmt_stat_level = 'L1,L1';

-- Clear the sql_patch and environment.
gaussdb=# SELECT dbe_sql_util.drop_sql_patch('patch1');
gaussdb=# DEALLOCATE ALL;

-- PRARARE
gaussdb=# PREPARE p1 AS SELECT * FROM rewrite_rule_hint_t1,(SELECT * FROM rewrite_rule_hint_t2
WHERE a > 1) tt WHERE rewrite_rule_hint_t1.a = tt.a;

-- View the unique SQL ID.
gaussdb=# SELECT unique_query_id,query FROM dbe_perf.statement_history WHERE query LIKE
'%rewrite_rule_hint%' ORDER BY finish_time DESC LIMIT 1;
 unique_query_id |
query
-----------------
+-----------------------------------------------------------------------------------------------------------------------------
---------------
       25719777 | prepare p1 as SELECT * FROM rewrite_rule_hint_t1,(SELECT * FROM rewrite_rule_hint_t2
WHERE a > 1) tt WHERE rewrite_rule_hint_t1.a = tt.a;

-- In this case, the unique SQL ID cannot make the SQL patch take effect.

-- Insert and analyze data to invalidate the cache.
gaussdb=# INSERT INTO rewrite_rule_hint_t1 VALUES(generate_series(1, 10000), generate_series(1, 10000),
generate_series(1, 10000),generate_series(1, 10000));
gaussdb=# ANALYZE rewrite_rule_hint_t1;

-- Generate a unique SQL ID again:
gaussdb=# EXPLAIN EXECUTE p1(1);
                                QUERY PLAN
-------------------------------------------------------------------------------
 Hash Join  (cost=39.62..277.01 rows=592 width=32)
   Hash Cond: (rewrite_rule_hint_t1.a = rewrite_rule_hint_t2.a)
   ->  Seq Scan on rewrite_rule_hint_t1  (cost=0.02..169.00 rows=9999 width=16)
         Filter: (a > 1)
   ->  Hash  (cost=32.20..32.20 rows=592 width=16)
         ->  Seq Scan on rewrite_rule_hint_t2  (cost=0.00..32.20 rows=592 width=16)
               Filter: (a > 1)
(7 rows)

-- Check the new unique SQL ID.
gaussdb=# SELECT unique_query_id,query FROM dbe_perf.statement_history WHERE query LIKE
'%rewrite_rule_hint%' ORDER BY finish_time DESC LIMIT 1;
 unique_query_id |
query
-----------------
+-----------------------------------------------------------------------------------------------------------------------------
---------------
     1116101547 | prepare p1 as SELECT * FROM rewrite_rule_hint_t1,(SELECT * FROM rewrite_rule_hint_t2
WHERE a > ?) tt WHERE rewrite_rule_hint_t1.a = tt.a;

The unique SQL ID changes and is generated using the SQL statement in PREPARE. In this case, the unique SQL ID is available.

-- Use SQL_PATCH.
gaussdb=# SELECT * FROM dbe_sql_util.create_hint_sql_patch('patch1', 1116101547 , 'set(enable_hashjoin
```



```
off) NO_EXPAND_SUBQUERY(@sel$2)');

-- Check whether it takes effect.
gaussdb=# EXPLAIN EXECUTE p1(1);
                                QUERY PLAN
-------------------------------------------------------------------------------------
 Merge Join  (cost=873.77..932.65 rows=592 width=32)
   Merge Cond: (rewrite_rule_hint_t1.a = rewrite_rule_hint_t2.a)
   ->  Sort  (cost=808.39..833.39 rows=10000 width=16)
         Sort Key: rewrite_rule_hint_t1.a
         ->  Seq Scan on rewrite_rule_hint_t1  (cost=0.00..144.00 rows=10000 width=16)
   ->  Sort  (cost=65.38..66.86 rows=592 width=16)
         Sort Key: rewrite_rule_hint_t2.a
         ->  Seq Scan on rewrite_rule_hint_t2  (cost=0.00..32.20 rows=592 width=16)
               Filter: (a > 1)
(9 rows)

The SQL patch takes effect and the corresponding plan is generated.
```

Helpful Links

For details about the system catalogs and functions related to the SQL patch, see **Table 1 System catalogs and functions related to SQL patches**.

Table 6-4 System catalogs and functions related to SQL patches

Name		Description
System catalog	**GS_SQL_PATCH**	GS_SQL_PATCH records the status information about all SQL patches.
Function **DBE_SQL_UTIL Schema**	**DBE_SQL_UTIL.create_hint_sql_patch**	create_hint_sql_patch creates hint SQL patches and returns whether the execution is successful.
	DBE_SQL_UTIL.create_abort_sql_patch	create_abort_sql_patch creates abort SQL patches and returns whether the execution is successful.
	DBE_SQL_UTIL.drop_sql_patch	drop_sql_patch deletes SQL patches from the connected CN and returns whether the execution is successful.
	DBE_SQL_UTIL.enable_sql_patch	enable_sql_patch enables SQL patches on the connected CN and returns whether the execution is successful.
	DBE_SQL_UTIL.disable_sql_patch	disable_sql_patch disables SQL patches on the connected CN and returns whether the execution is successful.
	DBE_SQL_UTIL.show_sql_patch	show_sql_patch displays the SQL patch corresponding to a specified patch name and return the running result.

Name		Description
	DBE_SQL_UTIL.create_hint_sql_patch	create_hint_sql_patch creates hint SQL patches and returns whether the execution is successful. This function is an overloaded function of the original function. The value of **parent_unique_sql_id** can be used to limit the effective range of the hint patch.
	DBE_SQL_UTIL.create_abort_sql_patch	create_abort_sql_patch creates abort SQL patches and returns whether the execution is successful. This function is an overloaded function of the original function. The value of **parent_unique_sql_id** can be used to limit the effective range of the abort patch.

6.12 Optimization Cases

6.12.1 Case: Modifying the GUC Parameter rewrite_rule

rewrite_rule contains multiple query rewriting rules: magicset, uniquecheck, intargetlist, and predpush. The following describes the application scenarios of some important rules.

Preparing the Case Environment

To demonstrate rule application scenarios, you need to prepare the following table creation statements:

```
-- Clean the environment.
DROP SCHEMA IF EXISTS rewrite_rule_guc_test CASCADE;
CREATE SCHEMA rewrite_rule_guc_test;
SET current_schema=rewrite_rule_guc_test;
-- Create a test table.
CREATE TABLE t(c1 INT, c2 INT, c3 INT, c4 INT);
CREATE TABLE t1(c1 INT, c2 INT, c3 INT, c4 INT);
CREATE TABLE t2(c1 INT, c2 INT, c3 INT, c4 INT);
```

intargetlist: Target Column Subquery Performance Improvement

The query performance can be greatly improved by converting the subquery in the target column to JOIN. The following is an example:

```
gaussdb=#  SET rewrite_rule='none';
SET
gaussdb=#  EXPLAIN (verbose on, costs off) SELECT c1,(SELECT avg(c2) FROM t2 WHERE t2.c2=t1.c2)
FROM t1 WHERE t1.c1<100 ORDER BY t1.c2;
                QUERY PLAN
-----------------------------------------------
 Sort
```



```
  Output: t1.c1, ((SubPlan 1)), t1.c2
  Sort Key: t1.c2
  ->  Seq Scan on public.t1
        Output: t1.c1, (SubPlan 1), t1.c2
        Filter: (t1.c1 < 100)
        SubPlan 1
          ->  Aggregate
                Output: avg(t2.c2)
                ->  Seq Scan on public.t2
                      Output: t2.c1, t2.c2
                      Filter: (t2.c2 = t1.c2)
(12 rows)
```

Because the subquery **(select avg(c2) from t2 where t2.c2=t1.c2)** in the target column cannot be pulled up, execution of the subquery is triggered each time a row of data of **t1** is scanned, and the query efficiency is low. If the **intargetlist** parameter is enabled, the subquery is converted to JOIN to improve the query performance.

```
gaussdb=#  SET rewrite_rule='intargetlist';
SET
gaussdb=# EXPLAIN (verbose on, costs off) SELECT c1,(SELECT avg(c2) FROM t2 WHERE t2.c2=t1.c2) FROM
t1 WHERE t1.c1<100 ORDER BY t1.c2;
                 QUERY PLAN
------------------------------------------------
 Sort
   Output: t1.c1, (avg(t2.c2)), t1.c2
   Sort Key: t1.c2
   ->  Hash Left Join
         Output: t1.c1, (avg(t2.c2)), t1.c2
         Hash Cond: (t1.c2 = t2.c2)
         ->  Seq Scan on public.t1
               Output: t1.c1, t1.c2
               Filter: (t1.c1 < 100)
         ->  Hash
               Output: (avg(t2.c2)), t2.c2
               ->  HashAggregate
                     Output: avg(t2.c2), t2.c2
                     Group By Key: t2.c2
                     ->  Seq Scan on public.t2
                           Output: t2.c2
(16 rows)
```

uniquecheck: Performance Improvement of Subqueries Without Aggregate Functions

To ensure subquery pullup, each condition must have only one line of output. The subqueries with aggregate functions can be automatically pulled up. For subqueries without aggregate functions, as shown in the following example:

```
SELECT t1.c1 FROM t1 WHERE t1.c1 = (SELECT t2.c1 FROM t2 WHERE t1.c1=t2.c2);
```

Rewrite as follows:

```
SELECT t1.c1 FROM t1 JOIN (SELECT t2.c1 FROM t2 WHERE t2.c1 IS NOT NULL GROUP BY t2.c1(unique
check)) tt(c1) on tt.c1=t1.c1;
```

Note that unique check in the preceding SQL statement indicates that **t2.c1** needs to be checked. If the SQL statement is abnormal, the SQL statement cannot be directly executed. To ensure semantic equivalence, the subquery **tt** must ensure that each **group by t2.c1** has only one line of output. Enable the **uniquecheck** query rewriting parameter to ensure that the query can be pulled up and equivalent. If more than one row of data is output at run time, an error is reported.


```
gaussdb=# SET rewrite_rule='uniquecheck';
SET
gaussdb=#  EXPLAIN verbose SELECT t1.c1 FROM t1 WHERE t1.c1 = (SELECT t2.c1 FROM t2 WHERE
t1.c1=t2.c1);
                                QUERY PLAN
-------------------------------------------------------------------------------
 Hash Join  (cost=43.36..104.40 rows=2149 distinct=[200, 200] width=4)
   Output: t1.c1
   Hash Cond: (t1.c1 = subquery."?column?")
   ->  Seq Scan on public.t1  (cost=0.00..31.49 rows=2149 width=4)
         Output: t1.c1, t1.c2
   ->  Hash  (cost=40.86..40.86 rows=200 width=8)
         Output: subquery."?column?", subquery.c1
         ->  Subquery Scan on subquery  (cost=36.86..40.86 rows=200 width=8)
               Output: subquery."?column?", subquery.c1
               ->  HashAggregate  (cost=36.86..38.86 rows=200 width=4)
                     Output: t2.c1, t2.c1
                     Group By Key: t2.c1
                     Filter: (t2.c1 IS NOT NULL)
                     Unique Check Required
                     ->  Seq Scan on public.t2  (cost=0.00..31.49 rows=2149 width=4)
                           Output: t2.c1
(16 rows)
```

Note: Because **group by t2.c1 unique check** occurs before the filter condition **tt.c1=t1.c1**, an error may be reported after the query that does not report an error is rewritten. An example is as follows:

There are tables **t1** and **t2**. The data in the tables is as follows:

```
gaussdb=#  SELECT * FROM t1 ORDER BY c2;
 c1 | c2
----+----
  1 |  1
  2 |  2
  3 |  3
(3 rows)
gaussdb=#  SELECT * FROM t2 ORDER BY c2;
 c1 | c2
----+----
  1 |  1
  2 |  2
  3 |  3
  4 |  4
  4 |  4
  5 |  5
(6 rows)
```

Disable and enable the **uniquecheck** parameter for comparison. After the parameter is enabled, an error is reported.

```
gaussdb=#  SELECT t1.c1 FROM t1 WHERE t1.c1 = (SELECT t2.c1 FROM t2 WHERE t1.c1=t2.c2) ;
 c1
----
  1
  2
  3
(3 rows)
gaussdb=#  SET rewrite_rule='uniquecheck';
SET
gaussdb=#  SELECT t1.c1 FROM t1 WHERE t1.c1 = (SELECT t2.c1 FROM t2 WHERE t1.c1=t2.c2) ;
ERROR:  more than one row returned by a subquery used as an expression
```

Parameter lazyagg: Eliminating the Aggregation Operation in a Subquery

The aggregation operation in a subquery is eliminated to improve the query efficiency. An example is as follows:


```
gaussdb=# SET rewrite_rule =none;
SET
gaussdb=# EXPLAIN (costs off) SELECT t.c2, sum(cc) FROM (SELECT c2, sum(c3) AS cc FROM t1 GROUP BY
c2) s1, t WHERE s1.c2=t.c2 GROUP BY t.c2 ORDER BY 1,2;
                    QUERY PLAN
-----------------------------------------------------
 Sort
   Sort Key: t.c2, (sum(s1.cc))
   ->  HashAggregate
         Group By Key: t.c2
         ->  Hash Join
               Hash Cond: (t.c2 = s1.c2)
               ->  Seq Scan on t
               ->  Hash
                     ->  Subquery Scan on s1
                           ->  HashAggregate
                                 Group By Key: t1.c2
                                 ->  Seq Scan on t1
(12 rows)
```

A subquery and the outer query have the same GROUP BY condition. The two-layer aggregation operation may lower down the query efficiency. Enable the **lazyagg** parameter to eliminate the aggregation operation in the subquery and improve the query performance.

```
gaussdb=# SET rewrite_rule = lazyagg;
SET
gaussdb=# EXPLAIN (costs off) SELECT t.c2, sum(cc) FROM (SELECT c2, sum(c3) AS cc FROM t1 GROUP BY
c2) s1, t WHERE s1.c2=t.c2 GROUP BY t.c2 ORDER BY 1,2;
                QUERY PLAN
-----------------------------------------
 Sort
   Sort Key: t.c2, (sum((t1.c3)::bigint))
   ->  HashAggregate
         Group By Key: t.c2
         ->  Hash Join
               Hash Cond: (t1.c2 = t.c2)
               ->  Seq Scan on t1
               ->  Hash
                     ->  Seq Scan on t
(9 rows)
```

Pushing Conditions from the Main Query to a Subquery

Join a subquery with the aggregation operator with the main query in advance. An example is as follows:

```
gaussdb=# SET rewrite_rule = none;
SET
gaussdb=# EXPLAIN (costs off) SELECT t1 FROM t1 WHERE t1.c2 = 10 AND t1.c3 < (SELECT sum(c3) FROM
t2 WHERE t1.c1 = t2.c1);
              QUERY PLAN
---------------------------------------
 Hash Join
   Hash Cond: (t2.c1 = t1.c1)
   Join Filter: (t1.c3 < (sum(t2.c3)))
   ->  HashAggregate
         Group By Key: t2.c1
         ->  Seq Scan on t2
   ->  Hash
         ->  Seq Scan on t1
               Filter: (c2 = 10)
(9 rows)
```

In this case, the join columns of the subquery are grouped and aggregated, and then the subquery is joined with the main query. This reduces repeated scanning

of related sublinks and improves query efficiency. After the rewriting parameters are modified, the plan is changed as follows:

```
gaussdb=# SET rewrite_rule = magicset;
SET
gaussdb=# explain (costs off) SELECT t1 FROM t1 WHERE t1.c2 = 10 AND t1.c3 < (SELECT sum(c3) FROM
t2 WHERE t1.c1 = t2.c1);
                        QUERY PLAN
------------------------------------------------------------------
 Hash Join
   Hash Cond: (t2.c1 = rewrite_rule_guc_test.t1.c1)
   Join Filter: (rewrite_rule_guc_test.t1.c3 < (sum(t2.c3)))
   ->  HashAggregate
         Group By Key: t2.c1
         ->  Hash Join
               Hash Cond: (t2.c1 = rewrite_rule_guc_test.t1.c1)
               ->  Seq Scan on t2
               ->  Hash
                     ->  HashAggregate
                           Group By Key: rewrite_rule_guc_test.t1.c1
                           ->  Seq Scan on t1
                                 Filter: (c2 = 10)
   ->  Hash
         ->  Seq Scan on t1
               Filter: (c2 = 10)
(16 rows)
```

6.12.2 Case: Creating an Appropriate Index

Symptom

Query the information about all personnel in the sales department.

```
-- Create a table.
CREATE TABLE staffs (staff_id NUMBER(6) NOT NULL, first_name VARCHAR2(20), last_name
VARCHAR2(25), employment_id VARCHAR2(10), section_id NUMBER(4), state_name VARCHAR2(10), city
VARCHAR2(10));
CREATE TABLE sections(section_id NUMBER(4), place_id NUMBER(4), section_name VARCHAR2(20));
CREATE TABLE states(state_id NUMBER(4));
CREATE TABLE places(place_id NUMBER(4), state_id  NUMBER(4));
-- Query before optimization.
EXPLAIN SELECT staff_id,first_name,last_name,employment_id,state_name,city
FROM staffs,sections,states,places
WHERE sections.section_name='Sales'
AND staffs.section_id = sections.section_id
AND sections.place_id = places.place_id
AND places.state_id = states.state_id
ORDER BY staff_id;
-- Query after optimization.
CREATE INDEX loc_id_pk ON places(place_id);
CREATE INDEX state_c_id_pk ON states(state_id);

EXPLAIN SELECT staff_id,first_name,last_name,employment_id,state_name,city
FROM staffs,sections,states,places
WHERE sections.section_name='Sales'
AND staffs.section_id = sections.section_id
AND sections.place_id = places.place_id
AND places.state_id = states.state_id
ORDER BY staff_id;
```

Optimization Analysis

The original execution plan is as follows before creating the **places.place_id** and **states.state_id** indexes:

```
                                QUERY PLAN
-----------------------------------------------------------------------------------------------
```



```
 Sort  (cost=125.25..126.34 rows=438 width=254)
   Sort Key: staffs.staff_id
   ->  Hash Join  (cost=64.91..106.03 rows=438 width=254)
         Hash Cond: (states.state_id = places.state_id)
         ->  Seq Scan on states  (cost=0.00..29.45 rows=1945 width=12)
         ->  Hash  (cost=64.35..64.35 rows=45 width=266)
               ->  Hash Join  (cost=33.09..64.35 rows=45 width=266)
                     Hash Cond: (places.place_id = sections.place_id)
                     ->  Seq Scan on places  (cost=0.00..25.13 rows=1513 width=24)
                     ->  Hash  (cost=33.02..33.02 rows=6 width=266)
                           ->  Hash Join  (cost=19.16..33.02 rows=6 width=266)
                                 Hash Cond: (staffs.section_id = sections.section_id)
                                 ->  Seq Scan on staffs  (cost=0.00..12.76 rows=276 width=266)
                                 ->  Hash  (cost=19.11..19.11 rows=4 width=24)
                                       ->  Seq Scan on sections  (cost=0.00..19.11 rows=4 width=24)
                                             Filter: ((section_name)::text = 'Sales'::text)
(16 rows)
```

The optimized execution plan is as follows (two indexes have been created on the **places.place_id** and **states.state_id** columns):

```
                                       QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Sort  (cost=107.40..108.49 rows=438 width=254)
   Sort Key: staffs.staff_id
   ->  Hash Join  (cost=35.37..88.18 rows=438 width=254)
         Hash Cond: (sections.section_id = staffs.section_id)
         ->  Nested Loop  (cost=19.16..66.85 rows=292 width=12)
               ->  Hash Join  (cost=19.16..50.27 rows=30 width=24)
                     Hash Cond: (places.place_id = sections.place_id)
                     ->  Seq Scan on places  (cost=0.00..25.13 rows=1513 width=24)
                     ->  Hash  (cost=19.11..19.11 rows=4 width=24)
                           ->  Seq Scan on sections  (cost=0.00..19.11 rows=4 width=24)
                                 Filter: ((section_name)::text = 'Sales'::text)
               ->  Index Only Scan using state_c_id_pk on states  (cost=0.00..0.45 rows=10 width=12)
                     Index Cond: (state_id = places.state_id)
         ->  Hash  (cost=12.76..12.76 rows=276 width=266)
               ->  Seq Scan on staffs  (cost=0.00..12.76 rows=276 width=266)
(15 rows)
```

6.12.3 Case: Adding NOT NULL for the JOIN Columns

```
SELECT * FROM join_a a JOIN join_b b ON a.b = b.b;
```

The execution plan is as follows:

```
                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=58.35..14677.69 rows=1074607 width=16) (actual time=23.374..23.384 rows=10 loops=1)
   Hash Cond: (a.b = b.b)
   ->  Seq Scan on join_a a  (cost=0.00..2248.10 rows=100010 width=8) (actual time=0.495..12.551
rows=100010 loops=1)
   ->  Hash  (cost=31.49..31.49 rows=2149 width=8) (actual time=0.614..0.614 rows=1000 loops=1)
         Buckets: 32768  Batches: 1  Memory Usage: 40kB
         ->  Seq Scan on join_b b  (cost=0.00..31.49 rows=2149 width=8) (actual time=0.009..0.183 rows=1000
loops=1)
 Total runtime: 23.716 ms
(7 rows)
```

Optimization Analysis

1. According to the execution plan, the sequential scan phase is time consuming.
2. Therefore, you are advised to manually add **NOT NULL** for the **JOIN** column in the statement as follows:

   ```
   SELECT
    *
   SELECT * FROM join_a a JOIN join_b b ON a.b = b.b where a.b IS NOT NULL;
   ```


The execution plan is as follows:

```
                                                    QUERY PLAN
----------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=58.22..14560.97 rows=1063762 width=16) (actual time=13.237..13.247 rows=10 
loops=1)
   Hash Cond: (a.b = b.b)
   ->  Seq Scan on join_a a  (cost=0.00..2248.10 rows=99510 width=8) (actual time=12.417..12.422 
rows=10 loops=1)
         Filter: (b IS NOT NULL)
         Rows Removed by Filter: 100000
   ->  Hash  (cost=31.49..31.49 rows=2138 width=8) (actual time=0.566..0.566 rows=1000 loops=1)
          Buckets: 32768  Batches: 1  Memory Usage: 40kB
         ->  Seq Scan on join_b b  (cost=0.00..31.49 rows=2138 width=8) (actual time=0.011..0.229 
rows=1000 loops=1)
               Filter: (b IS NOT NULL)
 Total runtime: 13.556 ms
(10 rows)
```

6.12.4 Case: Modifying a Partitioned Table

Symptom

In the following simple SQL statements, the performance bottlenecks exist in the scan operation on the **normal_date** table:

```
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
-----------------
 Seq Scan on normal_date  (cost=0.00..259.00 rows=30 width=12) (actual time=0.100..3.466 rows=30 
loops=1)
   Filter: (("time" >= '2022-09-01 00:00:00'::timestamp without time zone) AND ("time" <= '2022-10-01 
00:00:00'::timestamp without time zone))
   Rows Removed by Filter: 9970
 Total runtime: 3.587 ms
(4 rows)
```

Optimization Analysis

Obviously, there are date features in the **time** column of table data in the service layer, and this meet the features of a partitioned table. Replan the table definition of the **normal_date** table. Set the **time** column as a partition key, and month as an interval unit. Define the partitioned table **normal_date_part**. The modified result is as follows, and the performance is improved by nearly 10 times:

```
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
----------------------
 Partition Iterator  (cost=0.00..480.00 rows=30 width=12) (actual time=0.038..0.085 rows=30 loops=1)
   Iterations: 2
   ->  Partitioned Seq Scan on normal_date_part  (cost=0.00..480.00 rows=30 width=12) (actual 
time=0.049..0.063 rows=30 loops=2)
         Filter: (("time" >= '2022-09-01 00:00:00'::timestamp without time zone) AND ("time" <= '2022-10-01 
00:00:00'::timestamp without time zone))
         Rows Removed by Filter: 31
         Selected Partitions:  3..4
 Total runtime: 0.360 ms
(7 rows)
```

6.12.5 Case: Rewriting SQL Statements to Eliminate Subqueries

Symptom

```
select
    1,
    (select count(*) from normal_date n where n.id = a.id) as GZCS
from normal_date a;
```

This SQL performance is poor. SubPlan exists in the execution plan as follows:

```
                                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
---------
 Seq Scan on normal_date a  (cost=0.00..888118.42 rows=5129 width=4) (actual time=2.394..22194.907
rows=10000 loops=1)
   SubPlan 1
     ->  Aggregate  (cost=173.12..173.12 rows=1 width=8) (actual time=22179.496..22179.942 rows=10000
loops=10000)
           ->  Seq Scan on normal_date n  (cost=0.00..173.11 rows=1 width=0) (actual
time=11279.349..22159.608 rows=10000 loops=10000)
                 Filter: (id = a.id)
                 Rows Removed by Filter: 99990000
 Total runtime: 22196.415 ms
(7 rows)
```

Optimization

The core of this optimization is to eliminate subqueries. Based on the service scenario analysis, **a.id** is not NULL. In terms of SQL syntax, you can rewrite the SQL statement as follows:

```
select
count(*)
from normal_date n, normal_date a
where n.id = a.id
group by  a.id;
The plan is as follows:
                                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
----
 HashAggregate  (cost=480.86..532.15 rows=5129 width=12) (actual time=21.539..24.356 rows=10000
loops=1)
   Group By Key: a.id
   ->  Hash Join  (cost=224.40..455.22 rows=5129 width=4) (actual time=6.402..13.484 rows=10000 loops=1)
         Hash Cond: (n.id = a.id)
         ->  Seq Scan on normal_date n  (cost=0.00..160.29 rows=5129 width=4) (actual time=0.087..1.459
rows=10000 loops=1)
         ->  Hash  (cost=160.29..160.29 rows=5129 width=4) (actual time=6.065..6.065 rows=10000 loops=1)
               Buckets: 32768  Batches: 1  Memory Usage: 352kB
               ->  Seq Scan on normal_date a  (cost=0.00..160.29 rows=5129 width=4) (actual time=0.046..2.738
rows=10000 loops=1)
 Total runtime: 26.844 ms
(9 rows)
```

NOTE

To ensure that the modified statements have the same functions, NOT NULL is added to **normal_date.id**.

6.12.6 Case: Rewriting SQL Statements and Deleting in-clause

Symptom

in-clause/any-clause is a common SQL statement constraint. Sometimes, the clause following **in** or **any** is a constant. For example:

```
select
count(1)
from calc_empfyc_c1_result_tmp_t1
where ls_pid_cusr1 in ('20120405', '20130405');
```

Or:

```
select
count(1)
from calc_empfyc_c1_result_tmp_t1
where ls_pid_cusr1 in any('20120405', '20130405');
```

Sometimes, the **in** or **any** clause is used as follows:

```
SELECT
*
FROM test1 t1, test2 t2
WHERE t1.a = any(values(t2.a),(t2.b));
```

a and **b** are two columns in **t2**, and **"t1.a = any(values(t2.ba,(t2.b))"** is equivalent to **"t1.a = t2.a or t1.a = t2.b"**.

Therefore, join-condition is essentially an inequality, and nestloop must be used for this unequal join operation. The corresponding execution plan is as follows:

```
                                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
---
 Nested Loop  (cost=0.00..138614.38 rows=2309100 width=16) (actual time=0.152..19225.483 rows=1000
loops=1)
   Join Filter: (SubPlan 1)
   Rows Removed by Join Filter: 999000
   ->  Seq Scan on test1 t1  (cost=0.00..31.49 rows=2149 width=8) (actual time=0.021..3.309 rows=1000
loops=1)
   ->  Materialize  (cost=0.00..42.23 rows=2149 width=8) (actual time=0.331..1265.810 rows=1000000
loops=1000)
         ->  Seq Scan on test2 t2  (cost=0.00..31.49 rows=2149 width=8) (actual time=0.013..0.268 rows=1000
loops=1)
   SubPlan 1
     ->  Values Scan on "*VALUES*"  (cost=0.00..0.03 rows=2 width=4) (actual time=2890.741..7372.739
rows=1999000 loops=1000000)
 Total runtime: 19227.328 ms
(9 rows)
```

Optimization

The test result shows that both result sets are too large. As a result, nestloop is time-consuming with more than one hour to return results. Therefore, the key to performance optimization is to eliminate nestloop, using more efficient hash join. From the perspective of semantic equivalence, the SQL statements can be written as follows:

```
SELECT
*
FROM (
    SELECT * FROM test1 t1, test2 t2 WHERE t1.a = t2.a
    UNION
```

```
   SELECT * FROM test1 t1, test2 t2 WHERE t1.a = t2.b
);
```

The optimized SQL query consists of two equivalent join subqueries, and each subquery can be used for hash join in this scenario. The optimized execution plan is as follows:

```
                                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
---
 HashAggregate  (cost=1634.99..2096.81 rows=46182 width=16) (actual time=6.369..6.772 rows=1000
loops=1)
   Group By Key: t1.a, t1.b, t2.a, t2.b
   ->  Append  (cost=58.35..1173.17 rows=46182 width=16) (actual time=0.833..3.414 rows=2000 loops=1)
         ->  Hash Join  (cost=58.35..355.67 rows=23091 width=16) (actual time=0.832..1.590 rows=1000
loops=1)
               Hash Cond: (t1.a = t2.a)
               ->  Seq Scan on test1 t1  (cost=0.00..31.49 rows=2149 width=8) (actual time=0.015..0.156
rows=1000 loops=1)
               ->  Hash  (cost=31.49..31.49 rows=2149 width=8) (actual time=0.531..0.531 rows=1000 loops=1)
                      Buckets: 32768  Batches: 1  Memory Usage: 40kB
                     ->  Seq Scan on test2 t2  (cost=0.00..31.49 rows=2149 width=8) (actual time=0.010..0.199
rows=1000 loops=1)
         ->  Hash Join  (cost=58.35..355.67 rows=23091 width=16) (actual time=0.694..1.421 rows=1000
loops=1)
               Hash Cond: (t1.a = t2.b)
               ->  Seq Scan on test1 t1  (cost=0.00..31.49 rows=2149 width=8) (actual time=0.010..0.160
rows=1000 loops=1)
               ->  Hash  (cost=31.49..31.49 rows=2149 width=8) (actual time=0.524..0.524 rows=1000 loops=1)
                      Buckets: 32768  Batches: 1  Memory Usage: 40kB
                     ->  Seq Scan on test2 t2  (cost=0.00..31.49 rows=2149 width=8) (actual time=0.008..0.177
rows=1000 loops=1)
 Total runtime: 7.759 ms
(16 rows)
```

7 SQL Reference

7.1 GaussDB SQL

What Is SQL?

SQL is a standard computer language used to control the access to databases and manage data in databases.

SQL provides different statements to enable you to:

- Query data.
- Insert, update, and delete rows.
- Create, replace, modify, and delete objects.
- Control the access to a database and its objects.
- Maintain the consistency and integrity of a database.

SQL consists of commands and functions that are used to manage databases and database objects. SQL can also forcibly implement the rules for data types, expressions, and texts. Therefore, **SQL Reference** describes data types, expressions, functions, and operators in addition to SQL syntax.

Development of SQL Standards

Released SQL standards are as follows:

- 1986: ANSI X3.135-1986, ISO/IEC 9075:1986, SQL-86
- 1989: ANSI X3.135-1989, ISO/IEC 9075:1989, SQL-89
- 1992: ANSI X3.135-1992, ISO/IEC 9075:1992, SQL-92 (SQL2)
- 1999: ISO/IEC 9075:1999, SQL:1999 (SQL3)
- 2003: ISO/IEC 9075:2003, SQL:2003 (SQL4)
- 2011: ISO/IEC 9075:200N, SQL:2011 (SQL5)

SQL Standards Supported by GaussDB

By default, GaussDB supports most features of SQL5.

7.2 Keywords

SQL statements are classified into reserved keywords and non-reserved keywords. For details about common SQL keywords, see **Table 7-1**. Standards require that reserved keywords not be used as other identifiers. Non-reserved keywords have special meanings only in a specific environment and can be used as identifiers in other environments.

NOTICE

1. Currently, the non-reserved keywords have the following restrictions when being used as the identifier of a database object:
   1. It cannot be directly used as a column alias. That is, usage similar to SELECT 1 ABORT may cause errors.
   2. Keywords ENTITYESCAPING, NOENTITYESCAPING, and WELLFORMED cannot be used as identifiers of table names, column names, table aliases, column aliases, and function names if they are not enclosed in double quotation marks.
   3. The RAW keyword without double quotation marks cannot be used as the identifier of a table name or function name.
   4. The SET keyword without double quotation marks cannot be used as an identifier of a table alias. That is, usage similar to SELECT * FROM T1 SET may cause errors.
   5. Keywords such as BEGIN, BY, CLOSE, CURSOR, DECLARE, DELETE, EXECUTE, FUNCTION, IF, IMMEDIATE, INSERT, LOOP, MOVE, OF, REF, RELEASE, RETURN, SAVEPOINT, STRICT, TYPE, and UPDATE without double quotation marks cannot be used as variable names.
   6. When the SYS_REFCURSOR keyword is used as the identifier of a database object, if double quotation marks are not attached, a database object named **REFCURSOR** is created. If double quotation marks are attached, a database object named **SYS_REFCURSOR** is created.
2. Similar to the non-reserved keywords, the non-reserved (cannot be a function or type) keywords cannot be directly used as column aliases, either.
3. The reserved keyword CURRENT_TIMESTAMP with double quotation marks cannot be used as a function name.

Identifier Naming Conventions

Identifier naming must comply with the following rules:

- An identifier name can only contain letters, digits, underscores (_), and dollar signs ($).
- An identifier name must start with a letter or an underscore (_).

NOTE

- The naming rules are recommended but not required.
- In special cases, double quotation marks (") can be used to avoid special character errors.

SQL Keywords

Table 7-1 SQL keywords

Keyword	GaussDB	SQL:1999	SQL-92
ABORT	Non-reserved	N/A	N/A
ABS	N/A	Non-reserved	N/A
ABSOLUTE	Non-reserved	Reserved	Reserved
ACCESS	Non-reserved	N/A	N/A
ACCOUNT	Non-reserved	N/A	N/A
ACTION	Non-reserved	Reserved	Reserved
ADA	N/A	Non-reserved	Non-reserved
ADD	Non-reserved	Reserved	Reserved
ADDDATE	Non-reserved	N/A	N/A
ADMIN	Non-reserved	Reserved	N/A
ADVANCED	Non-reserved	N/A	N/A
AFTER	Non-reserved	Reserved	N/A
AGGREGATE	Non-reserved	Reserved	N/A
ALGORITHM	Non-reserved	N/A	N/A
ALIAS	N/A	Reserved	N/A
ALL	Reserved	Reserved	Reserved
ALLOCATE	N/A	Reserved	Reserved
ALSO	Non-reserved	N/A	N/A
ALTER	Non-reserved	Reserved	Reserved
ALWAYS	Non-reserved	N/A	N/A
ANALYSE	Reserved	N/A	N/A
ANALYZE	Reserved	N/A	N/A
AND	Reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
ANY	Reserved	Reserved	Reserved
APP	Non-reserved	N/A	N/A
APPEND	Non-reserved	N/A	N/A
ARCHIVE	Non-reserved	N/A	N/A
ARE	N/A	Reserved	Reserved
ARRAY	Reserved	Reserved	N/A
AS	Reserved	Reserved	Reserved
ASC	Reserved	Reserved	Reserved
ASENSITIVE	N/A	Non-reserved	N/A
ASSERTION	Non-reserved	Reserved	Reserved
ASSIGNMENT	Non-reserved	Non-reserved	N/A
ASYMMETRIC	Reserved	Non-reserved	N/A
AT	Non-reserved	Reserved	Reserved
ATOMIC	N/A	Non-reserved	N/A
ATTRIBUTE	Non-reserved	N/A	N/A
AUDIT	Non-reserved	N/A	N/A
AUTHID	Reserved	N/A	N/A
AUTHORIZATION	Reserved (functions and types allowed)	Reserved	Reserved
AUTO	Non-reserved	N/A	N/A
AUTO_INCREMENT	Non-reserved	N/A	N/A
AUTOEXTEND	Non-reserved	N/A	N/A
AUTOMAPPED	Non-reserved	N/A	N/A
AUTOMATIC	Non-reserved	N/A	N/A
AVG	N/A	Non-reserved	Reserved
BACKWARD	Non-reserved	N/A	N/A
BAD_PATH	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
BARRIER	Non-reserved	N/A	N/A
BEFORE	Non-reserved	Reserved	N/A
BEGIN	Non-reserved	Reserved	Reserved
BEGIN_NON_ANOYBLOCK	Non-reserved	N/A	N/A
BETWEEN	Non-reserved (cannot be functions or types)	Non-reserved	Reserved
BIGINT	Non-reserved (cannot be functions or types)	N/A	N/A
BINARY	Reserved (functions and types allowed)	Reserved	N/A
BINARY_DOUBLE	Non-reserved (cannot be functions or types)	N/A	N/A
BINARY_INTEGER	Non-reserved (cannot be functions or types)	N/A	N/A
BIT	Non-reserved (cannot be functions or types)	Reserved	Reserved
BIT_LENGTH	N/A	Non-reserved	Reserved
BITVAR	N/A	Non-reserved	N/A
BLANKS	Non-reserved	N/A	N/A
BLOB	Non-reserved	Reserved	N/A
BLOCKCHAIN	Non-reserved	N/A	N/A
BODY	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
BOOLEAN	Non-reserved (cannot be functions or types)	Reserved	N/A
BOTH	Reserved	Reserved	Reserved
BREADTH	N/A	Reserved	N/A
BUCKETS	Reserved	N/A	N/A
BY	Non-reserved	Reserved	Reserved
BYTEAWITHOUTORDER	Non-reserved (cannot be functions or types)	N/A	N/A
BYTEAWITHOUTORDER-WITHEQUAL	Non-reserved (cannot be functions or types)	N/A	N/A
C	N/A	Non-reserved	Non-reserved
CACHE	Non-reserved	N/A	N/A
CALL	Non-reserved	Reserved	N/A
CALLED	Non-reserved	Non-reserved	N/A
CANCELABLE	Non-reserved	N/A	N/A
CARDINALITY	N/A	Non-reserved	N/A
CASCADE	Non-reserved	Reserved	Reserved
CASCADED	Non-reserved	Reserved	Reserved
CASE	Reserved	Reserved	Reserved
CAST	Reserved	Reserved	Reserved
CATALOG	Non-reserved	Reserved	Reserved
CATALOG_NAME	N/A	Non-reserved	Non-reserved
CHAIN	Non-reserved	Non-reserved	N/A
CHANGE	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
CHAR	Non-reserved (cannot be functions or types)	Reserved	Reserved
CHAR_LENGTH	N/A	Non-reserved	Reserved
CHARACTER	Non-reserved (cannot be functions or types)	Reserved	Reserved
CHARACTER_LENGTH	N/A	Non-reserved	Reserved
CHARACTER_SET_CATALOG	N/A	Non-reserved	Non-reserved
CHARACTER_SET_NAME	N/A	Non-reserved	Non-reserved
CHARACTER_SET_SCHEMA	N/A	Non-reserved	Non-reserved
CHARACTERISTICS	Non-reserved	N/A	N/A
CHARACTERSET	Non-reserved	N/A	N/A
CHARSET	Non-reserved	N/A	N/A
CHECK	Reserved	Reserved	Reserved
CHECKED	N/A	Non-reserved	N/A
CHECKPOINT	Non-reserved	N/A	N/A
CLASS	Non-reserved	Reserved	N/A
CLASS_ORIGIN	N/A	Non-reserved	Non-reserved
CLEAN	Non-reserved	N/A	N/A
CLIENT	Non-reserved	N/A	N/A
CLIENT_MASTER_KEY	Non-reserved	N/A	N/A
CLIENT_MASTER_KEYS	Non-reserved	N/A	N/A
CLOB	Non-reserved	Reserved	N/A
CLOSE	Non-reserved	Reserved	Reserved
CLUSTER	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
COALESCE	Non-reserved (cannot be functions or types)	Non-reserved	Reserved
COBOL	N/A	Non-reserved	Non-reserved
COLLATE	Reserved	Reserved	Reserved
COLLATION	Reserved (functions and types allowed)	Reserved	Reserved
COLLATION_CATALOG	N/A	Non-reserved	Non-reserved
COLLATION_NAME	N/A	Non-reserved	Non-reserved
COLLATION_SCHEMA	N/A	Non-reserved	Non-reserved
COLUMN	Reserved	Reserved	Reserved
COLUMN_ENCRYPTION_KEY	Non-reserved	N/A	N/A
COLUMN_ENCRYPTION_KEYS	Non-reserved	N/A	N/A
COLUMN_NAME	N/A	Non-reserved	Non-reserved
COLUMNS	Non-reserved	N/A	N/A
COMMAND_FUNCTION	N/A	Non-reserved	Non-reserved
COMMAND_FUNCTION_CODE	N/A	Non-reserved	N/A
COMMENT	Non-reserved	N/A	N/A
COMMENTS	Non-reserved	N/A	N/A
COMMIT	Non-reserved	Reserved	Reserved
COMMITTED	Non-reserved	Non-reserved	Non-reserved
COMPACT	Reserved (functions and types allowed)	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
COMPATIBLE_ILLEGAL_CHARS	Non-reserved	N/A	N/A
COMPILE	Non-reserved	N/A	N/A
COMPLETE	Non-reserved	N/A	N/A
COMPLETION	Non-reserved	Reserved	N/A
COMPRESS	Non-reserved	N/A	N/A
CONCURRENTLY	Reserved (functions and types allowed)	N/A	N/A
CONDITION	Non-reserved	N/A	N/A
CONDITION_NUMBER	N/A	Non-reserved	Non-reserved
CONFIGURATION	Non-reserved	N/A	N/A
CONNECT	Non-reserved	Reserved	Reserved
CONNECTION	Non-reserved	Reserved	Reserved
CONNECTION_NAME	N/A	Non-reserved	Non-reserved
CONSTANT	Non-reserved	N/A	N/A
CONSTRAINT	Reserved	Reserved	Reserved
CONSTRAINT_CATALOG	N/A	Non-reserved	Non-reserved
CONSTRAINT_NAME	N/A	Non-reserved	Non-reserved
CONSTRAINT_SCHEMA	N/A	Non-reserved	Non-reserved
CONSTRAINTS	Non-reserved	Reserved	Reserved
CONSTRUCTOR	N/A	Reserved	N/A
CONTAINING	Non-reserved	N/A	N/A
CONTAINS	N/A	Non-reserved	N/A
CONTENT	Non-reserved	N/A	N/A
CONTINUE	Non-reserved	Reserved	Reserved
CONVERSION	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
CONVERT	Non-reserved	Non-reserved	Reserved
COORDINATOR	Non-reserved	N/A	N/A
COORDINATORS	Non-reserved	N/A	N/A
COPY	Non-reserved	N/A	N/A
CORRESPONDING	N/A	Reserved	Reserved
COST	Non-reserved	N/A	N/A
COUNT	N/A	Non-reserved	Reserved
CREATE	Reserved	Reserved	Reserved
CROSS	Reserved (functions and types allowed)	Reserved	Reserved
CROSSBUCKET	Reserved	N/A	N/A
CSN	Reserved (functions and types allowed)	N/A	N/A
CSV	Non-reserved	N/A	N/A
CUBE	Non-reserved	Reserved	N/A
CURRENT	Non-reserved	Reserved	Reserved
CURRENT_CATALOG	Reserved	N/A	N/A
CURRENT_DATE	Reserved	Reserved	Reserved
CURRENT_PATH	N/A	Reserved	N/A
CURRENT_ROLE	Reserved	Reserved	N/A
CURRENT_SCHEMA	Reserved (functions and types allowed)	N/A	N/A
CURRENT_TIME	Reserved	Reserved	Reserved
CURRENT_TIMESTAMP	Reserved	Reserved	Reserved
CURRENT_USER	Reserved	Reserved	Reserved
CURSOR	Non-reserved	Reserved	Reserved
CURSOR_NAME	N/A	Non-reserved	Non-reserved
CYCLE	Non-reserved	Reserved	N/A

Keyword	GaussDB	SQL:1999	SQL-92
DATA	Non-reserved	Reserved	Non-reserved
DATABASE	Non-reserved	N/A	N/A
DATAFILE	Non-reserved	N/A	N/A
DATANODE	Non-reserved	N/A	N/A
DATANODES	Non-reserved	N/A	N/A
DATATYPE_CL	Non-reserved	N/A	N/A
DATE	Non-reserved (cannot be functions or types)	Reserved	Reserved
DATE_ADD	Non-reserved	N/A	N/A
DATE_FORMAT	Non-reserved	N/A	N/A
DATE_SUB	Non-reserved	N/A	N/A
DATETIME	Non-reserved	N/A	N/A
DATETIME_INTERVAL_CODE	N/A	Non-reserved	Non-reserved
DATETIME_INTERVAL_PRECISION	N/A	Non-reserved	Non-reserved
DAY	Non-reserved	Reserved	Reserved
DAY_HOUR	Non-reserved	N/A	N/A
DAY_MICROSECOND	Non-reserved	N/A	N/A
DAY_MINUTE	Non-reserved	N/A	N/A
DAY_SECOND	Non-reserved	N/A	N/A
DAYS	Non-reserved	N/A	N/A
DB4AISHOT	Non-reserved	N/A	N/A
DBCOMPATIBILITY	Non-reserved	N/A	N/A
DBTIMEZONE	Reserved	N/A	N/A
DEALLOCATE	Non-reserved	Reserved	Reserved
DEC	Non-reserved (cannot be functions or types)	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
DECIMAL	Non-reserved (cannot be functions or types)	Reserved	Reserved
DECLARE	Non-reserved	Reserved	Reserved
DECODE	Non-reserved (cannot be functions or types)	N/A	N/A
DEFAULT	Reserved	Reserved	Reserved
DEFAULTS	Non-reserved	N/A	N/A
DEFERRABLE	Reserved	Reserved	Reserved
DEFERRED	Non-reserved	Reserved	Reserved
DEFINED	N/A	Non-reserved	N/A
DEFINER	Non-reserved	Non-reserved	N/A
DELETE	Non-reserved	Reserved	Reserved
DELETE_ALL	Non-reserved	N/A	N/A
DELIMITER	Non-reserved	N/A	N/A
DELIMITERS	Non-reserved	N/A	N/A
DELTA	Non-reserved	N/A	N/A
DELTAMERGE	Reserved (functions and types allowed)	N/A	N/A
DEPTH	N/A	Reserved	N/A
DEREF	N/A	Reserved	N/A
DESC	Reserved	Reserved	Reserved
DESCRIBE	N/A	Reserved	Reserved
DESCRIPTOR	N/A	Reserved	Reserved
DESTROY	N/A	Reserved	N/A
DESTRUCTOR	N/A	Reserved	N/A
DETERMINISTIC	Non-reserved	Reserved	N/A
DIAGNOSTICS	N/A	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
DICTIONARY	Non-reserved	Reserved	N/A
DIRECT	Non-reserved	N/A	N/A
DIRECTORY	Non-reserved	N/A	N/A
DISABLE	Non-reserved	N/A	N/A
DISABLE_ALL	Non-reserved	N/A	N/A
DISCARD	Non-reserved	N/A	N/A
DISCARD_PATH	Non-reserved	N/A	N/A
DISCONNECT	Non-reserved	Reserved	Reserved
DISPATCH	N/A	Non-reserved	N/A
DISTINCT	Reserved	Reserved	Reserved
DISTRIBUTE	Non-reserved	N/A	N/A
DISTRIBUTED	Non-reserved	N/A	N/A
DISTRIBUTION	Non-reserved	N/A	N/A
DO	Reserved	N/A	N/A
DOCUMENT	Non-reserved	N/A	N/A
DOMAIN	Non-reserved	Reserved	Reserved
DOUBLE	Non-reserved	Reserved	Reserved
DROP	Non-reserved	Reserved	Reserved
DUMPFILE	Non-reserved	N/A	N/A
DUPLICATE	Non-reserved	N/A	N/A
DYNAMIC	N/A	Reserved	N/A
DYNAMIC_FUNCTION	N/A	Non-reserved	Non-reserved
DYNAMIC_FUNCTION_CODE	N/A	Non-reserved	N/A
EACH	Non-reserved	Reserved	N/A
ELASTIC	Non-reserved	N/A	N/A
ELSE	Reserved	Reserved	Reserved
ENABLE	Non-reserved	N/A	N/A
ENABLE_ALL	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
ENCLOSED	Non-reserved	N/A	N/A
ENCODING	Non-reserved	N/A	N/A
ENCRYPTED	Non-reserved	N/A	N/A
ENCRYPTED_VALUE	Non-reserved	N/A	N/A
ENCRYPTION	Non-reserved	N/A	N/A
ENCRYPTION_TYPE	Non-reserved	N/A	N/A
END	Reserved	Reserved	Reserved
END-EXEC	N/A	Reserved	Reserved
ENDS	Non-reserved	N/A	N/A
ENFORCED	Non-reserved	N/A	N/A
ENGINE	Non-reserved	N/A	N/A
ENTITYESCAPING	Non-reserved	N/A	N/A
ENUM	Non-reserved	N/A	N/A
EOL	Non-reserved	N/A	N/A
EQUALS	N/A	Reserved	N/A
ERROR	Non-reserved	N/A	N/A
ERRORS	Non-reserved	N/A	N/A
ESCAPE	Non-reserved	Reserved	Reserved
ESCAPED	Non-reserved	N/A	N/A
ESCAPING	Non-reserved	N/A	N/A
EVALNAME	Non-reserved	N/A	N/A
EVENT	Non-reserved	N/A	N/A
EVENTS	Non-reserved	N/A	N/A
EVERY	Non-reserved	Reserved	N/A
EXCEPT	Reserved	Reserved	Reserved
EXCEPTION	N/A	Reserved	Reserved
EXCHANGE	Non-reserved	N/A	N/A
EXCLUDE	Non-reserved	N/A	N/A
EXCLUDED	Reserved	N/A	N/A
EXCLUDING	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
EXCLUSIVE	Non-reserved	N/A	N/A
EXEC	N/A	Reserved	Reserved
EXECUTE	Non-reserved	Reserved	Reserved
EXISTING	N/A	Non-reserved	N/A
EXISTS	Non-reserved (cannot be functions or types)	Non-reserved	Reserved
EXPDP	Non-reserved	N/A	N/A
EXPIRED	Non-reserved	N/A	N/A
EXPLAIN	Non-reserved	N/A	N/A
EXTEND	Non-reserved	N/A	N/A
EXTENSION	Non-reserved	N/A	N/A
EXTERNAL	Non-reserved	Reserved	Reserved
EXTRACT	Non-reserved (cannot be functions or types)	Non-reserved	Reserved
FALSE	Reserved	Reserved	Reserved
FAMILY	Non-reserved	N/A	N/A
FAST	Non-reserved	N/A	N/A
FEATURES	Non-reserved	N/A	N/A
FENCED	Reserved	N/A	N/A
FETCH	Reserved	Reserved	Reserved
FIELDS	Non-reserved	N/A	N/A
FILEHEADER	Non-reserved	N/A	N/A
FILL_MISSING_FIELDS	Non-reserved	N/A	N/A
FILLER	Non-reserved	N/A	N/A
FILTER	Non-reserved	N/A	Reserved
FINAL	N/A	Non-reserved	N/A
FINISH	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
FIRST	Non-reserved	Reserved	Reserved
FIXED	Non-reserved	N/A	Reserved
FLOAT	Non-reserved (cannot be functions or types)	Reserved	Reserved
FOLLOWING	Non-reserved	N/A	N/A
FOR	Reserved	Reserved	Reserved
FORCE	Non-reserved	N/A	N/A
FOREIGN	Reserved	Reserved	Reserved
FORMATTER	Non-reserved	N/A	N/A
FORTRAN	N/A	Non-reserved	Non-reserved
FORWARD	Non-reserved	N/A	N/A
FOUND	N/A	Reserved	Reserved
FREE	N/A	Reserved	N/A
FREEZE	Reserved (functions and types allowed)	N/A	N/A
FROM	Reserved	Reserved	Reserved
FULL	Reserved (functions and types allowed)	Reserved	Reserved
FUNCTION	Non-reserved	Reserved	N/A
FUNCTIONS	Non-reserved	N/A	N/A
G	N/A	Non-reserved	N/A
GENERAL	N/A	Reserved	N/A
GENERATED	Non-reserved	Non-reserved	N/A
GET	N/A	Reserved	Reserved
GET_FORMAT	Non-reserved	N/A	N/A
GLOBAL	Non-reserved	Reserved	Reserved
GO	N/A	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
GOTO	N/A	Reserved	Reserved
GRANT	Reserved	Reserved	Reserved
GRANTED	Non-reserved	Non-reserved	N/A
GREATEST	Non-reserved (cannot be functions or types)	N/A	N/A
GROUP	Reserved	Reserved	Reserved
GROUPING	Non-reserved (cannot be functions or types)	Reserved	N/A
GSIUSABLE	Non-reserved	N/A	N/A
GSIVALID	Non-reserved	N/A	N/A
GSIWAITALL	Non-reserved	N/A	N/A
HANDLER	Non-reserved	N/A	N/A
HAVING	Reserved	Reserved	Reserved
HDFSDIRECTORY	Reserved (functions and types allowed)	N/A	N/A
HEADER	Non-reserved	N/A	N/A
HIERARCHY	N/A	Non-reserved	N/A
HOLD	Non-reserved	Non-reserved	N/A
HOST	N/A	Reserved	N/A
HOUR	Non-reserved	Reserved	Reserved
HOUR_MICROSECOND	Non-reserved	N/A	N/A
HOUR_MINUTE	Non-reserved	N/A	N/A
HOUR_SECOND	Non-reserved	N/A	N/A
IDENTIFIED	Non-reserved	N/A	N/A
IDENTITY	Non-reserved	Reserved	Reserved
IF	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
IFNULL	Non-reserved	N/A	N/A
IGNORE	Non-reserved	Reserved	N/A
IGNORE_EXTRA_DATA	Non-reserved	N/A	N/A
ILIKE	Reserved (functions and types allowed)	N/A	N/A
ILM	Non-reserved	N/A	N/A
ILM_PIDX_LIST	Non-reserved	N/A	N/A
IMMEDIATE	Non-reserved	Reserved	Reserved
IMMUTABLE	Non-reserved	N/A	N/A
IMPDP	Non-reserved	N/A	N/A
IMPLEMENTATION	N/A	Non-reserved	N/A
IMPLICIT	Non-reserved	N/A	N/A
IN	Reserved	Reserved	Reserved
INCLUDE	Non-reserved	N/A	N/A
INCLUDING	Non-reserved	N/A	N/A
INCREMENT	Non-reserved	N/A	N/A
INCREMENTAL	Non-reserved	N/A	N/A
INDEX	Non-reserved	N/A	N/A
INDEXES	Non-reserved	N/A	N/A
INDICATOR	N/A	Reserved	Reserved
INFILE	Non-reserved	N/A	N/A
INFIX	N/A	Non-reserved	N/A
INHERIT	Non-reserved	N/A	N/A
INHERITS	Non-reserved	N/A	N/A
INITIAL	Non-reserved	N/A	N/A
INITIALIZE	N/A	Reserved	N/A
INITIALLY	Reserved	Reserved	Reserved
INITRANS	Non-reserved	N/A	N/A
INLINE	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
INNER	Reserved (functions and types allowed)	Reserved	Reserved
INOUT	Non-reserved (cannot be functions or types)	Reserved	N/A
INPUT	Non-reserved	Reserved	Reserved
INSENSITIVE	Non-reserved	Non-reserved	Reserved
INSERT	Non-reserved	Reserved	Reserved
INSTANCE	N/A	Non-reserved	N/A
INSTANTIABLE	N/A	Non-reserved	N/A
INSTEAD	Non-reserved	N/A	N/A
INT	Non-reserved (cannot be functions or types)	Reserved	Reserved
INTEGER	Non-reserved (cannot be functions or types)	Reserved	Reserved
INTERNAL	Non-reserved	N/A	N/A
INTERSECT	Reserved	Reserved	Reserved
INTERVAL	Non-reserved (cannot be functions or types)	Reserved	Reserved
INTO	Reserved	Reserved	Reserved
INVISIBLE	Non-reserved	N/A	N/A
INVOKER	Non-reserved	Non-reserved	N/A
IP	Non-reserved	N/A	N/A
IS	Reserved	Reserved	Reserved
ISNULL	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
ISOLATION	Non-reserved	Reserved	Reserved
ITERATE	N/A	Reserved	N/A
JOIN	Reserved (functions and types allowed)	Reserved	Reserved
JSON_OBJECT	Non-reserved	N/A	N/A
K	N/A	Non-reserved	N/A
KEY	Non-reserved	Reserved	Reserved
KEY_MEMBER	N/A	Non-reserved	N/A
KEY_PATH	Non-reserved	N/A	N/A
KEY_STORE	Non-reserved	N/A	N/A
KEY_TYPE	N/A	Non-reserved	N/A
KILL	Non-reserved	N/A	N/A
LABEL	Non-reserved	N/A	N/A
LANGUAGE	Non-reserved	Reserved	Reserved
LARGE	Non-reserved	Reserved	N/A
LAST	Non-reserved	Reserved	Reserved
LAST_DAY	Non-reserved	N/A	N/A
LATERAL	N/A	Reserved	N/A
LC_COLLATE	Non-reserved	N/A	N/A
LC_CTYPE	Non-reserved	N/A	N/A
LEADING	Reserved	Reserved	Reserved
LEAKPROOF	Non-reserved	N/A	N/A
LEAST	Non-reserved (cannot be functions or types)	N/A	N/A
LEFT	Reserved (functions and types allowed)	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
LENGTH	N/A	Non-reserved	Non-reserved
LESS	Reserved	Reserved	N/A
LEVEL	Non-reserved	Reserved	Reserved
LIKE	Reserved (functions and types allowed)	Reserved	Reserved
LIMIT	Reserved	Reserved	N/A
LINES	Non-reserved	N/A	N/A
LINK	Non-reserved	N/A	N/A
LIST	Non-reserved	N/A	N/A
LISTEN	Non-reserved	N/A	N/A
LNNVL	Non-reserved (cannot be functions or types)	N/A	N/A
LOAD	Non-reserved	N/A	N/A
LOAD_BAD	Non-reserved	N/A	N/A
LOAD_DISCARD	Non-reserved	N/A	N/A
LOAD_INTEGER_LEN	Non-reserved	N/A	N/A
LOAD_SESSION_ID	Non-reserved	N/A	N/A
LOAD_SMALLINT_LEN	Non-reserved	N/A	N/A
LOAD_UNFIXED_LEN	Non-reserved	N/A	N/A
LOAD_UNFIXED_START_POS	Non-reserved	N/A	N/A
LOCAL	Non-reserved	Reserved	Reserved
LOCALTIME	Reserved	Reserved	N/A
LOCALTIMESTAMP	Reserved	Reserved	N/A
LOCATION	Non-reserved	N/A	N/A
LOCATOR	N/A	Reserved	N/A
LOCK	Non-reserved	N/A	N/A
LOCKED	Non-reserved	N/A	N/A
LOG	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
LOGGING	Non-reserved	N/A	N/A
LOGIN_ANY	Non-reserved	N/A	N/A
LOGIN_FAILURE	Non-reserved	N/A	N/A
LOGIN_SUCCESS	Non-reserved	N/A	N/A
LOGOUT	Non-reserved	N/A	N/A
LOOP	Non-reserved	N/A	N/A
LOWER	N/A	Non-reserved	Reserved
M	N/A	Non-reserved	N/A
MANUAL	Non-reserved	N/A	N/A
MAP	N/A	Reserved	N/A
MAPPING	Non-reserved	N/A	N/A
MARK	Non-reserved	N/A	N/A
MASKING	Non-reserved	N/A	N/A
MASTER	Non-reserved	N/A	N/A
MATCH	Non-reserved	Reserved	Reserved
MATCHED	Non-reserved	N/A	N/A
MATERIALIZED	Non-reserved	N/A	N/A
MAX	N/A	Non-reserved	Reserved
MAXEXTENTS	Non-reserved	N/A	N/A
MAXSIZE	Non-reserved	N/A	N/A
MAXTRANS	Non-reserved	N/A	N/A
MAXVALUE	Reserved	N/A	N/A
MEDIUMINT	Non-reserved (cannot be functions or types)	N/A	N/A
MERGE	Non-reserved	N/A	N/A
MESSAGE_LENGTH	N/A	Non-reserved	Non-reserved

Keyword	GaussDB	SQL:1999	SQL-92
MESSAGE_OCTET_LENGTH	N/A	Non-reserved	Non-reserved
MESSAGE_TEXT	N/A	Non-reserved	Non-reserved
METHOD	N/A	Non-reserved	N/A
MICROSECOND	Non-reserved	N/A	N/A
MIN	N/A	Non-reserved	Reserved
MINEXTENTS	Non-reserved	N/A	N/A
MINUS	Reserved	N/A	N/A
MINUTE	Non-reserved	Reserved	Reserved
MINUTE_MICROSECOND	Non-reserved	N/A	N/A
MINUTE_SECOND	Non-reserved	N/A	N/A
MINVALUE	Non-reserved	N/A	N/A
MOD	N/A	Non-reserved	N/A
MODE	Non-reserved	N/A	N/A
MODEL	Non-reserved	N/A	N/A
MODIFICATION	Non-reserved	N/A	N/A
MODIFIES	N/A	Reserved	N/A
MODIFY	Reserved	Reserved	N/A
MODULE	N/A	Reserved	Reserved
MONTH	Non-reserved	Reserved	Reserved
MONTHS	Non-reserved	N/A	N/A
MORE	N/A	Non-reserved	Non-reserved
MOVE	Non-reserved	N/A	N/A
MOVEMENT	Non-reserved	N/A	N/A
MUMPS	N/A	Non-reserved	Non-reserved
NAME	Non-reserved	Non-reserved	Non-reserved

Keyword	GaussDB	SQL:1999	SQL-92
NAMES	Non-reserved	Reserved	Reserved
NATIONAL	Non-reserved (cannot be functions or types)	Reserved	Reserved
NATURAL	Reserved (functions and types allowed)	Reserved	Reserved
NCHAR	Non-reserved (cannot be functions or types)	Reserved	Reserved
NCLOB	N/A	Reserved	N/A
NEW	N/A	Reserved	N/A
NEXT	Non-reserved	Reserved	Reserved
NO	Non-reserved	Reserved	Reserved
NOCACHE	Non-reserved	N/A	N/A
NOCOMPRESS	Non-reserved	N/A	N/A
NOCYCLE	Reserved	N/A	N/A
NODE	Non-reserved	N/A	N/A
NOENTITYESCAPING	Non-reserved	N/A	N/A
NOEXTEND	Non-reserved	N/A	N/A
NOLOGGING	Non-reserved	N/A	N/A
NOMAXVALUE	Non-reserved	N/A	N/A
NOMINVALUE	Non-reserved	N/A	N/A
NONE	Non-reserved (cannot be functions or types)	Reserved	N/A
NOSCALE	Non-reserved	N/A	N/A
NOT	Reserved	Reserved	Reserved
NOTHING	Non-reserved	N/A	N/A
NOTIFY	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
NOTNULL	Reserved (functions and types allowed)	N/A	N/A
NOW	Non-reserved	N/A	N/A
NOWAIT	Non-reserved	N/A	N/A
NULL	Reserved	Reserved	Reserved
NULLABLE	N/A	Non-reserved	Non-reserved
NULLCOLS	Non-reserved	N/A	N/A
NULLIF	Non-reserved (cannot be functions or types)	Non-reserved	Reserved
NULLS	Non-reserved	N/A	N/A
NUMBER	Non-reserved (cannot be functions or types)	Non-reserved	Non-reserved
NUMERIC	Non-reserved (cannot be functions or types)	Reserved	Reserved
NUMSTR	Non-reserved	N/A	N/A
NVARCHAR2	Non-reserved (cannot be functions or types)	N/A	N/A
NVL	Non-reserved (cannot be functions or types)	N/A	N/A
NVL2	Non-reserved (cannot be functions or types)	N/A	N/A
OBJECT	Non-reserved	Reserved	N/A
OCTET_LENGTH	N/A	Non-reserved	Reserved
OF	Non-reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
OFF	Non-reserved	Reserved	N/A
OFFSET	Reserved	N/A	N/A
OIDS	Non-reserved	N/A	N/A
OLD	N/A	Reserved	N/A
ON	Reserved	Reserved	Reserved
ONLY	Reserved	Reserved	Reserved
OPEN	N/A	Reserved	Reserved
OPERATION	N/A	Reserved	N/A
OPERATOR	Non-reserved	N/A	N/A
OPTIMIZATION	Non-reserved	N/A	N/A
OPTION	Non-reserved	Reserved	Reserved
OPTIONALLY	Non-reserved	N/A	N/A
OPTIONS	Non-reserved	Non-reserved	N/A
OR	Reserved	Reserved	Reserved
ORDER	Reserved	Reserved	Reserved
ORDINALITY	Non-reserved	Reserved	N/A
OUT	Non-reserved (cannot be functions or types)	Reserved	N/A
OUTER	Reserved (functions and types allowed)	Reserved	Reserved
OUTFILE	Non-reserved	N/A	N/A
OUTPUT	N/A	Reserved	Reserved
OVER	Non-reserved	N/A	N/A
OVERLAPS	Reserved (functions and types allowed)	Non-reserved	Reserved
OVERLAY	Non-reserved (cannot be functions or types)	Non-reserved	N/A

Keyword	GaussDB	SQL:1999	SQL-92
OVERRIDING	N/A	Non-reserved	N/A
OWNED	Non-reserved	N/A	N/A
OWNER	Non-reserved	N/A	N/A
PACKAGE	Non-reserved	N/A	N/A
PACKAGES	Non-reserved	N/A	N/A
PAD	N/A	Reserved	Reserved
PARAMETER	N/A	Reserved	N/A
PARAMETER_MODE	N/A	Non-reserved	N/A
PARAMETER_NAME	N/A	Non-reserved	N/A
PARAMETER_ORDINAL_POSITION	N/A	Non-reserved	N/A
PARAMETER_SPECIFIC_CATALOG	N/A	Non-reserved	N/A
PARAMETER_SPECIFIC_NAME	N/A	Non-reserved	N/A
PARAMETER_SPECIFIC_SCHEMA	N/A	Non-reserved	N/A
PARAMETERS	N/A	Reserved	N/A
PARSER	Non-reserved	N/A	N/A
PARTIAL	Non-reserved	Reserved	Reserved
PARTITION	Non-reserved	N/A	N/A
PARTITIONING	Non-reserved	N/A	N/A
PARTITIONS	Non-reserved	N/A	N/A
PASCAL	N/A	Non-reserved	Non-reserved
PASSING	Non-reserved	N/A	N/A
PASSWORD	Non-reserved	N/A	N/A
PATH	N/A	Reserved	N/A
PCTFREE	Non-reserved	N/A	N/A
PER	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
PERCENT	Non-reserved	N/A	N/A
PERFORMANCE	Reserved	N/A	N/A
PERM	Non-reserved	N/A	N/A
PIVOT	Non-reserved	N/A	N/A
PLACING	Reserved	N/A	N/A
PLAN	Non-reserved	N/A	N/A
PLANS	Non-reserved	N/A	N/A
PLI	N/A	Non-reserved	Non-reserved
POLICY	Non-reserved	N/A	N/A
POOL	Non-reserved	N/A	N/A
POSITION	Non-reserved (cannot be functions or types)	Non-reserved	Reserved
POSTFIX	N/A	Reserved	N/A
PRECEDING	Non-reserved	N/A	N/A
PRECISION	Non-reserved (cannot be functions or types)	Reserved	Reserved
PREDICT	Non-reserved	N/A	N/A
PREFERRED	Non-reserved	N/A	N/A
PREFIX	Non-reserved	Reserved	N/A
PREORDER	N/A	Reserved	N/A
PREPARE	Non-reserved	Reserved	Reserved
PREPARED	Non-reserved	N/A	N/A
PRESERVE	Non-reserved	Reserved	Reserved
PRIMARY	Reserved	Reserved	Reserved
PRIOR	Non-reserved	Reserved	Reserved
PRIORER	Reserved	N/A	N/A
PRIVATE	Non-reserved	N/A	N/A
PRIVILEGE	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
PRIVILEGES	Non-reserved	Reserved	Reserved
PROCEDURAL	Non-reserved	N/A	N/A
PROCEDURE	Reserved	Reserved	Reserved
PROFILE	Non-reserved	N/A	N/A
PUBLIC	Non-reserved	Reserved	Reserved
PUBLISH	Non-reserved	N/A	N/A
PURGE	Non-reserved	N/A	N/A
QUARTER	Non-reserved	N/A	N/A
QUERY	Non-reserved	N/A	N/A
QUOTE	Non-reserved	N/A	N/A
RANDOMIZED	Non-reserved	N/A	N/A
RANGE	Non-reserved	N/A	N/A
RATIO	Non-reserved	N/A	N/A
RAW	Non-reserved	N/A	N/A
READ	Non-reserved	Reserved	Reserved
READS	N/A	Reserved	N/A
REAL	Non-reserved (cannot be functions or types)	Reserved	Reserved
REASSIGN	Non-reserved	N/A	N/A
REBUILD	Non-reserved	N/A	N/A
RECHECK	Non-reserved	N/A	N/A
RECOVER	Non-reserved	N/A	N/A
RECURSIVE	Non-reserved	Reserved	N/A
RECYCLEBIN	Reserved (functions and types allowed)	N/A	N/A
REDISANYVALUE	Non-reserved	N/A	N/A
REF	Non-reserved	Reserved	N/A
REFERENCES	Reserved	Reserved	Reserved
REFERENCING	N/A	Reserved	N/A

Keyword	GaussDB	SQL:1999	SQL-92
REFRESH	Non-reserved	N/A	N/A
REGEXP	Reserved (functions and types allowed)	N/A	N/A
REGEXP_LIKE	Non-reserved (cannot be functions or types)	N/A	N/A
REINDEX	Non-reserved	N/A	N/A
REJECT	Reserved	N/A	N/A
RELATIVE	Non-reserved	Reserved	Reserved
RELEASE	Non-reserved	N/A	N/A
RELOPTIONS	Non-reserved	N/A	N/A
REMOTE	Non-reserved	N/A	N/A
REMOVE	Non-reserved	N/A	N/A
RENAME	Non-reserved	N/A	N/A
REPEATABLE	Non-reserved	Non-reserved	Non-reserved
REPLACE	Non-reserved	N/A	N/A
REPLICA	Non-reserved	N/A	N/A
RESET	Non-reserved	N/A	N/A
RESIZE	Non-reserved	N/A	N/A
RESOURCE	Non-reserved	N/A	N/A
RESPECT	Non-reserved	N/A	N/A
RESTART	Non-reserved	N/A	N/A
RESTRICT	Non-reserved	Reserved	Reserved
RESULT	N/A	Reserved	N/A
RETURN	Non-reserved	Reserved	N/A
RETURNED_LENGTH	N/A	Non-reserved	Non-reserved
RETURNED_OCTET_LENGTH	N/A	Non-reserved	Non-reserved
RETURNED_SQLSTATE	N/A	Non-reserved	Non-reserved

Keyword	GaussDB	SQL:1999	SQL-92
RETURNING	Reserved	N/A	N/A
RETURNS	Non-reserved	Reserved	N/A
REUSE	Non-reserved	N/A	N/A
REVOKE	Non-reserved	Reserved	Reserved
RIGHT	Reserved (functions and types allowed)	Reserved	Reserved
RLIKE	Reserved (functions and types allowed)	N/A	N/A
ROLE	Non-reserved	Reserved	N/A
ROLES	Non-reserved	N/A	N/A
ROLLBACK	Non-reserved	Reserved	Reserved
ROLLUP	Non-reserved	Reserved	N/A
ROTATION	Non-reserved	N/A	N/A
ROUTINE	N/A	Reserved	N/A
ROUTINE_CATALOG	N/A	Non-reserved	N/A
ROUTINE_NAME	N/A	Non-reserved	N/A
ROUTINE_SCHEMA	N/A	Non-reserved	N/A
ROW	Non-reserved (cannot be functions or types)	Reserved	N/A
ROW_COUNT	N/A	Non-reserved	Non-reserved
ROWNUM	Reserved	N/A	N/A
ROWS	Non-reserved	Reserved	Reserved
ROWTYPE	Non-reserved	N/A	N/A
RULE	Non-reserved	N/A	N/A
SAMPLE	Non-reserved	N/A	N/A
SAVEPOINT	Non-reserved	Reserved	N/A

Keyword	GaussDB	SQL:1999	SQL-92
SCALE	Non-reserved	Non-reserved	Non-reserved
SCHEDULE	Non-reserved	N/A	N/A
SCHEMA	Non-reserved	Reserved	Reserved
SCHEMA_NAME	N/A	Non-reserved	Non-reserved
SCOPE	N/A	Reserved	N/A
SCROLL	Non-reserved	Reserved	Reserved
SEARCH	Non-reserved	Reserved	N/A
SECOND	Non-reserved	Reserved	Reserved
SECOND_MICROSECOND	Non-reserved	N/A	N/A
SECTION	N/A	Reserved	Reserved
SECURITY	Non-reserved	Non-reserved	N/A
SELECT	Reserved	Reserved	Reserved
SELF	N/A	Non-reserved	N/A
SENSITIVE	N/A	Non-reserved	N/A
SEPARATOR	Non-reserved	N/A	N/A
SEQUENCE	Non-reserved	Reserved	N/A
SEQUENCES	Non-reserved	N/A	N/A
SERIALIZABLE	Non-reserved	Non-reserved	Non-reserved
SERVER	Non-reserved	N/A	N/A
SERVER_NAME	N/A	Non-reserved	Non-reserved
SESSION	Non-reserved	Reserved	Reserved
SESSION_USER	Reserved	Reserved	Reserved
SESSIONTIMEZONE	Reserved	N/A	N/A
SET	Non-reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
SETOF	Non-reserved (cannot be functions or types)	N/A	N/A
SETS	Non-reserved	Reserved	N/A
SHARE	Non-reserved	N/A	N/A
SHIPPABLE	Non-reserved	N/A	N/A
SHOW	Non-reserved	N/A	N/A
SHRINK	Reserved	N/A	N/A
SHUTDOWN	Non-reserved	N/A	N/A
SIBLINGS	Non-reserved	N/A	N/A
SIGNED	Non-reserved (cannot be functions or types)	N/A	N/A
SIMILAR	Reserved (functions and types allowed)	Non-reserved	N/A
SIMPLE	Non-reserved	Non-reserved	N/A
SIZE	Non-reserved	Reserved	Reserved
SKIP	Non-reserved	N/A	N/A
SLAVE	Non-reserved	N/A	N/A
SLICE	Non-reserved	N/A	N/A
SMALLDATETIME	Non-reserved (cannot be functions or types)	N/A	N/A
SMALLDATETIME_FORMAT	Non-reserved	N/A	N/A
SMALLINT	Non-reserved (cannot be functions or types)	Reserved	Reserved
SNAPSHOT	Non-reserved	N/A	N/A
SOME	Reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
SOURCE	Non-reserved	Non-reserved	N/A
SPACE	Non-reserved	Reserved	Reserved
SPECIFIC	N/A	Reserved	N/A
SPECIFIC_NAME	N/A	Non-reserved	N/A
SPECIFICATION	Non-reserved	N/A	N/A
SPECIFICTYPE	N/A	Reserved	N/A
SPILL	Non-reserved	N/A	N/A
SPLIT	Non-reserved	N/A	N/A
SQL	N/A	Reserved	Reserved
SQLCODE	N/A	N/A	Reserved
SQLERROR	N/A	N/A	Reserved
SQLEXCEPTION	N/A	Reserved	N/A
SQLSTATE	N/A	Reserved	Reserved
SQLWARNING	N/A	Reserved	N/A
STABLE	Non-reserved	N/A	N/A
STANDALONE	Non-reserved	N/A	N/A
START	Non-reserved	Reserved	N/A
STARTING	Non-reserved	N/A	N/A
STARTS	Non-reserved	N/A	N/A
STATE	N/A	Reserved	N/A
STATEMENT	Non-reserved	Reserved	N/A
STATEMENT_ID	Non-reserved	N/A	N/A
STATIC	N/A	Reserved	N/A
STATISTICS	Non-reserved	N/A	N/A
STDIN	Non-reserved	N/A	N/A
STDOUT	Non-reserved	N/A	N/A
STORAGE	Non-reserved	N/A	N/A
STORE	Non-reserved	N/A	N/A
STORED	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
STRATIFY	Non-reserved	N/A	N/A
STRICT	Non-reserved	N/A	N/A
STRIP	Non-reserved	N/A	N/A
STRUCTURE	N/A	Reserved	N/A
STYLE	N/A	Non-reserved	N/A
SUBCLASS_ORIGIN	N/A	Non-reserved	Non-reserved
SUBDATE	Non-reserved	N/A	N/A
SUBLIST	N/A	Non-reserved	N/A
SUBPARTITION	Non-reserved	N/A	N/A
SUBPARTITIONING	Non-reserved	N/A	N/A
SUBPARTITIONS	Non-reserved	N/A	N/A
SUBSTR	Non-reserved	N/A	N/A
SUBSTRING	Non-reserved (cannot be functions or types)	Non-reserved	Reserved
SUM	N/A	Non-reserved	Reserved
SYMMETRIC	Reserved	Non-reserved	N/A
SYNONYM	Non-reserved	N/A	N/A
SYS_REFCURSOR	Non-reserved	N/A	N/A
SYSDATE	Reserved	N/A	N/A
SYSID	Non-reserved	N/A	N/A
SYSTEM	Non-reserved	Non-reserved	N/A
SYSTEM_USER	N/A	Reserved	Reserved
TABLE	Reserved	Reserved	Reserved
TABLE_NAME	N/A	Non-reserved	Non-reserved
TABLES	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
TABLESAMPLE	Reserved (functions and types allowed)	N/A	N/A
TABLESPACE	Non-reserved	N/A	N/A
TARGET	Non-reserved	N/A	N/A
TEMP	Non-reserved	N/A	N/A
TEMPLATE	Non-reserved	N/A	N/A
TEMPORARY	Non-reserved	Reserved	Reserved
TERMINATE	N/A	Reserved	N/A
TERMINATED	Non-reserved	N/A	N/A
TEXT	Non-reserved	N/A	N/A
THAN	Non-reserved	Reserved	N/A
THEN	Reserved	Reserved	Reserved
TIME	Non-reserved (cannot be functions or types)	Reserved	Reserved
TIME_FORMAT	Non-reserved	N/A	N/A
TIMECAPSULE	Reserved (functions and types allowed)	N/A	N/A
TIMESTAMP	Non-reserved (cannot be functions or types)	Reserved	Reserved
TIMESTAMP_FORMAT	Non-reserved	N/A	N/A
TIMESTAMPADD	Non-reserved	N/A	N/A
TIMESTAMPDIFF	Non-reserved (cannot be functions or types)	N/A	N/A
TIMEZONE_HOUR	N/A	Reserved	Reserved
TIMEZONE_MINUTE	N/A	Reserved	Reserved
TINYINT	Non-reserved (cannot be functions or types)	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
TO	Reserved	Reserved	Reserved
TRAILING	Reserved	Reserved	Reserved
TRANSACTION	Non-reserved	Reserved	Reserved
TRANSACTION_ACTIVE	N/A	Non-reserved	N/A
TRANSACTIONS_COMMITTED	N/A	Non-reserved	N/A
TRANSACTIONS_ROLLED_BACK	N/A	Non-reserved	N/A
TRANSFORM	Non-reserved	Non-reserved	N/A
TRANSFORMS	N/A	Non-reserved	N/A
TRANSLATE	N/A	Non-reserved	Reserved
TRANSLATION	N/A	Reserved	Reserved
TREAT	Non-reserved (cannot be functions or types)	Reserved	N/A
TRIGGER	Non-reserved	Reserved	N/A
TRIGGER_CATALOG	N/A	Non-reserved	N/A
TRIGGER_NAME	N/A	Non-reserved	N/A
TRIGGER_SCHEMA	N/A	Non-reserved	N/A
TRIM	Non-reserved (cannot be functions or types)	Non-reserved	Reserved
TRUE	Reserved	Reserved	Reserved
TRUNCATE	Non-reserved	N/A	N/A
TRUSTED	Non-reserved	N/A	N/A
TSFIELD	Non-reserved	N/A	N/A
TSTAG	Non-reserved	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
TSTIME	Non-reserved	N/A	N/A
TYPE	Non-reserved	Non-reserved	Non-reserved
TYPES	Non-reserved	N/A	N/A
UNBOUNDED	Non-reserved	N/A	N/A
UNCOMMITTED	Non-reserved	Non-reserved	Non-reserved
UNDER	N/A	Reserved	N/A
UNENCRYPTED	Non-reserved	N/A	N/A
UNION	Reserved	Reserved	Reserved
UNIQUE	Reserved	Reserved	Reserved
UNKNOWN	Non-reserved	Reserved	Reserved
UNLIMITED	Non-reserved	N/A	N/A
UNLISTEN	Non-reserved	N/A	N/A
UNLOCK	Non-reserved	N/A	N/A
UNLOGGED	Non-reserved	N/A	N/A
UNNAMED	N/A	Non-reserved	Non-reserved
UNNEST	N/A	Reserved	N/A
UNPIVOT	Non-reserved	N/A	N/A
UNSIGNED	Non-reserved (cannot be functions or types)	N/A	N/A
UNTIL	Non-reserved	N/A	N/A
UNUSABLE	Non-reserved	N/A	N/A
UPDATE	Non-reserved	Reserved	Reserved
UPPER	N/A	Non-reserved	Reserved
USAGE	N/A	Reserved	Reserved
USEEOF	Non-reserved	N/A	N/A
USER	Reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
USER_DEFINED_TYPE_CATALOG	N/A	Non-reserved	N/A
USER_DEFINED_TYPE_NAME	N/A	Non-reserved	N/A
USER_DEFINED_TYPE_SCHEMA	N/A	Non-reserved	N/A
USING	Reserved	Reserved	Reserved
VACUUM	Non-reserved	N/A	N/A
VALID	Non-reserved	N/A	N/A
VALIDATE	Non-reserved	N/A	N/A
VALIDATION	Non-reserved	N/A	N/A
VALIDATOR	Non-reserved	N/A	N/A
VALUE	Non-reserved	Reserved	Reserved
VALUES	Non-reserved (cannot be functions or types)	Reserved	Reserved
VARCHAR	Non-reserved (cannot be functions or types)	Reserved	Reserved
VARCHAR2	Non-reserved (cannot be functions or types)	N/A	N/A
VARIABLE	N/A	Reserved	N/A
VARIABLES	Non-reserved	N/A	N/A
VARIADIC	Reserved	N/A	N/A
VARYING	Non-reserved	Reserved	Reserved
VCGROUP	Non-reserved	N/A	N/A
VERBOSE	Reserved (functions and types allowed)	N/A	N/A
VERIFY	Reserved	N/A	N/A
VERSION	Non-reserved	N/A	N/A
VIEW	Non-reserved	Reserved	Reserved

Keyword	GaussDB	SQL:1999	SQL-92
VISIBLE	Non-reserved	N/A	N/A
VOLATILE	Non-reserved	N/A	N/A
WAIT	Non-reserved	N/A	N/A
WEAK	Non-reserved	N/A	N/A
WEEK	Non-reserved	N/A	N/A
WELLFORMED	Non-reserved	N/A	N/A
WHEN	Reserved	Reserved	Reserved
WHENEVER	N/A	Reserved	Reserved
WHERE	Reserved	Reserved	Reserved
WHITESPACE	Non-reserved	N/A	N/A
WINDOW	Reserved	N/A	N/A
WITH	Reserved	Reserved	Reserved
WITHIN	Non-reserved	N/A	N/A
WITHOUT	Non-reserved	Reserved	N/A
WORK	Non-reserved	Reserved	Reserved
WORKLOAD	Non-reserved	N/A	N/A
WRAPPER	Non-reserved	N/A	N/A
WRITE	Non-reserved	Reserved	Reserved
XML	Non-reserved	N/A	N/A
XMLATTRIBUTES	Non-reserved (cannot be functions or types)	N/A	N/A
XMLCONCAT	Non-reserved (cannot be functions or types)	N/A	N/A
XMLELEMENT	Non-reserved (cannot be functions or types)	N/A	N/A
XMLEXISTS	Non-reserved (cannot be functions or types)	N/A	N/A

Keyword	GaussDB	SQL:1999	SQL-92
XMLFOREST	Non-reserved (cannot be functions or types)	N/A	N/A
XMLNAMESPACES	Non-reserved	N/A	N/A
XMLPARSE	Non-reserved (cannot be functions or types)	N/A	N/A
XMLPI	Non-reserved (cannot be functions or types)	N/A	N/A
XMLROOT	Non-reserved (cannot be functions or types)	N/A	N/A
XMLSERIALIZE	Non-reserved (cannot be functions or types)	N/A	N/A
XMLTABLE	Non-reserved	N/A	N/A
XMLTYPE	Non-reserved (cannot be functions or types)	N/A	N/A
YEAR	Non-reserved	Reserved	Reserved
YEAR_MONTH	Non-reserved	N/A	N/A
YEARS	Non-reserved	N/A	N/A
YES	Non-reserved	N/A	N/A
ZEROFILL	Non-reserved (cannot be functions or types)	N/A	N/A
ZONE	Non-reserved	Reserved	Reserved

Columns listed in the following table cannot be used as column names during table creation.

CTID	XMIN	CMIN	XMAX	CMAX
TABLEOID	XC_NODE_ID	XC_NODE_HASH	N/A	GS_TUPLE_UID
TABLEBUCKETID	N/A	N/A	N/A	N/A

7.3 Data Types

A data type is a basic data attribute. Occupied storage space and allowed operations vary according to data types. Data is stored in tables in a database. Each column of a data table specifies a data type and data must be stored according to data types.

GaussDB supports implicit conversions between certain data types. For details, see **PG_CAST**.

7.3.1 Numeric Types

Table 7-2 lists all available types. For arithmetic operators and related built-in functions, see **Arithmetic Functions and Operators**.

Table 7-2 Integer types

Name	Description	Storage Space	Range
TINYINT [UNSIGNED]	Tiny integer. The signed alias is INT1, and the unsigned alias is UINT1.	1 byte	• The signed value ranges from 0 to +255. • The unsigned value ranges from 0 to +255. • When **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, the signed value ranges from –128 to +127.
SMALLINT [UNSIGNED]	Small integer. The signed alias is INT2, and the unsigned alias is UINT2.	2 bytes	• The signed value ranges from –32,768 to +32,767. • The unsigned value ranges from 0 to +65,535.

Name	Description	Storage Space	Range
MEDIUMINT [UNSIGNED]	Medium-sized integer. The signed alias is INT4, and the unsigned alias is UINT4.	4 bytes	• The signed value ranges from –2147483648 to +2147483647. • The unsigned value ranges from 0 to +4294967295.
INTEGER [UNSIGNED]	Common integer. The signed alias is INT4, and the unsigned alias is UINT4.	4 bytes	• The signed value ranges from –2,147,483,648 to +2,147,483,647. • The unsigned value ranges from 0 to +4,294,967,295.
BINARY_INTEGER	Common integer, alias of signed INTEGER, compatible with database A.	4 bytes	–2,147,483,648 to +2,147,483,647
BIGINT [UNSIGNED]	Large integer. The signed alias is INT8, and the unsigned alias is UINT8.	8 bytes	• The signed value ranges from –9,223,372,036,854,775,808 to +9,223,372,036,854,775,807. • The unsigned value ranges from 0 to +18,446,744,073,709,551,615.
INT16 [SIGNED]	A 16-byte integer cannot be used to create tables.	16 bytes	–170,141,183,460,469,231,731,687,303,715,884,105,728 to +170,141,183,460,469,231,731,687,303,715,884,105,727

NOTE

- When **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, integers support the display width and **ZEROFILL** attributes.
- The display width does not limit the range of values that can be stored in a column, nor does it prevent values wider than the column display width from being displayed correctly. For example, a column specified as SMALLINT(3) has the usual SMALLINT range of –32768 to 32767, and values outside the range permitted by three digits are displayed in full using more than three digits.
- When the display width is used in conjunction with the **ZEROFILL** attribute, zeros are padded in front of the numeric value to achieve the display width. For example, for a column declared as **INT(4) ZEROFILL**, a value of **5** is retrieved as 0005.
- If you specify **ZEROFILL** for a numeric column, the **UNSIGNED** attribute is automatically added.
- If **ZEROFILL** is not specified for a numeric column and only the width information is specified, the width information is not displayed in the table structure description.

Example:

```
-- Create a database.
gaussdb=# CREATE DATABASE b_database dbcompatibility = 'B';
gaussdb=# \c b_database
-- Create a table containing TINYINT data.
b_database=# CREATE TABLE int_type_t1
         (
          IT_COL1 TINYINT,
          IT_COL2 TINYINT UNSIGNED
         );

-- Insert data to the created table.
b_database=# INSERT INTO int_type_t1 VALUES(10,20);

-- View data.
b_database=# SELECT * FROM int_type_t1;
it_col1 | it_col2
--------+---------
     10 |      20
(1 row)

-- Drop the table.
b_database=# DROP TABLE int_type_t1;
-- Create a table containing TINYINT, INTEGER, and BIGINT data.
b_database=# CREATE TABLE int_type_t2
(
    a TINYINT,
    b TINYINT,
    c INTEGER,
    d INTEGER UNSIGNED,
    e BIGINT,
    f BIGINT UNSIGNED
);

-- Insert data.
b_database=# INSERT INTO int_type_t2 VALUES(100, 10, 1000, 10000, 200, 2000);

-- View data.
b_database=# SELECT * FROM int_type_t2;
  a  | b  |  c   |   d    |  e  |  f
-----+----+------+--------+-----+------
 100 | 10 | 1000 |  10000 | 200 | 2000
(1 row)

-- Drop the table.
b_database=# DROP TABLE int_type_t2;
```



```
-- Delete the database.
b_database=# \c postgres
gaussdb=# DROP DATABASE b_database;
```

NOTE

- Numbers of the TINYINT, SMALLINT, INTEGER, BIGINT, or INT16 type, that is, integers can be stored. Saving a number with a decimal in any of the data types will result in errors.
- If **UNSIGNED** is specified, negative values are not allowed.
- The INTEGER type is the common choice, as it offers the best balance between range, storage size, and performance. Generally, use the SMALLINT type only if you are sure that the value range is within the SMALLINT value range. The storage speed of INTEGER is much faster. BIGINT is used only when the range of INTEGER is not large enough.
- The unsigned numeric type can be used only in the row-store engine when **sql_compatibility** is set to **'B'**.
- When a minus sign, plus sign, or multiplication sign is used between integer values (one of which is of the UNSIGNED type), the result is unsigned.
- The return value of the addition (+), subtraction (–), and multiplication (x) operations of the INT1, UINT1, UINT2, UINT4, or UINT8 type can exceed the range of the corresponding type. The return value of the addition (+), subtraction (–), and multiplication (x) operations of the INT2, INT4, or INT8 type cannot exceed the range of the corresponding type.
- The UNSIGNED type cannot be converted to the SET type. Do not calculate or compare the UNSIGNED type with the SET type.
- When **sql_compatibility** is set to **'B'**, non-numeric characters are automatically truncated or the value **0** is returned.

Table 7-3 Arbitrary precision types

Name	Description	Storage Space	Range
NUMERIC[(p[,s])], DECIMAL[(p[,s])]	• The value range of **p** is [1,1000], and the value range of **s** is [0,*p*]. • When **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, if the precision and scale are not specified, the default precision **p** is **10** and the scale **s** is **0**. If a NUMERIC type is not attached with parentheses, it is set to NUMERIC(10,0) only when being applied to table columns. In other scenarios, the NUMERIC type is used based on the original range by default. **NOTE** **p** indicates the total digits, and **s** indicates the decimal digit.	The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data.	Up to 131,072 digits before the decimal point, and up to 16,383 digits after the decimal point when no precision is specified.

Name	Description	Storage Space	Range
NUMBER[(p[,s])]	Alias of the NUMERIC type.	The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data.	Up to 131,072 digits before the decimal point, and up to 16,383 digits after the decimal point when no precision is specified.

Example:

```
-- Create a table.
gaussdb=# CREATE TABLE decimal_type_t1
(
    DT_COL1 DECIMAL(10,4)
);

-- Insert data.
gaussdb=# INSERT INTO decimal_type_t1 VALUES(123456.122331);

-- Query data in the table.
gaussdb=# SELECT * FROM decimal_type_t1;
   dt_col1
-------------
 123456.1223
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE decimal_type_t1;
-- Create a table.
gaussdb=# CREATE TABLE numeric_type_t1
(
    NT_COL1 NUMERIC(10,4)
);

-- Insert data.
gaussdb=# INSERT INTO numeric_type_t1 VALUES(123456.12354);

-- Query data in the table.
gaussdb=# SELECT * FROM numeric_type_t1;
   nt_col1
-------------
 123456.1235
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE numeric_type_t1;
```


NOTE

- Compared to the integer types, the arbitrary precision numbers require larger storage space and have lower storage efficiency, operation efficiency, and poorer compression ratio results. The INTEGER type is the common choice when number types are defined. Arbitrary precision numbers are used only when numbers exceed the maximum range indicated by the integers.
- When NUMERIC/DECIMAL is used for defining a column, you are advised to specify the precision (p) and scale (s) for the column.
- When **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, the input and output range specifications of the NUMERIC[(p[,s])] and DECIMAL[(p[,s])] types are different.
  - Input format: The value can be a number or a numeric string.
    - If the value of **SQL_MODE** contains **strict_trans_tables**, an error is reported if the input value is invalid or exceeds the range.
    - If the value of **SQL_MODE** does not contain **strict_trans_tables** and the input value is invalid or exceeds the range, a warning message is reported and **0** or the truncated value is returned.
  - Output format: The output format of the DECIMAL[(p[,s])] type is different from that of the NUMERIC[(p[,s])] type when the precision and scale are not specified. If NUMERIC has no precision or scale, the decimal part is retained during type conversion. For details, see the output format in the scenario where no parameter is set. If DECIMAL has no precision or scale, the precision is **10** and the scale is **0** during type conversion.

Table 7-4 Sequence integer

Name	Description	Storage Space	Range
SMALLSERIAL	Two-byte serial integer	2 bytes.	–32,768 to +32,767.
SERIAL	Four-byte serial integer	4 bytes.	–2,147,483,648 to +2,147,483,647.
BIGSERIAL	Eight-byte serial integer	8 bytes.	–9,223,372,036,854,775,808 to +9,223,372,036,854,775,807.
LARGESERIAL	By default, a 16-byte auto-incrementing integer is inserted. The actual value type is the same as that of NUMERIC.	Variable-length type. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data.	There can be a maximum of 131072 digits before the decimal point and 16383 digits after the decimal point.

Example:

```
-- Create a table.
gaussdb=# CREATE TABLE smallserial_type_tab(a SMALLSERIAL);
```



```
-- Insert data.
gaussdb=# INSERT INTO smallserial_type_tab VALUES(default);

-- Insert data again.
gaussdb=# INSERT INTO smallserial_type_tab VALUES(default);

-- View data.
gaussdb=# SELECT * FROM smallserial_type_tab;
 a
---
 1
 2
(2 rows)

-- Create a table.
gaussdb=# CREATE TABLE serial_type_tab(b SERIAL);

-- Insert data.
gaussdb=# INSERT INTO serial_type_tab VALUES(default);

-- Insert data again.
gaussdb=# INSERT INTO serial_type_tab VALUES(default);

-- View data.
gaussdb=# SELECT * FROM serial_type_tab;
 b
---
 1
 2
(2 rows)

-- Create a table.
gaussdb=# CREATE TABLE bigserial_type_tab(c BIGSERIAL);

-- Insert data.
gaussdb=# INSERT INTO bigserial_type_tab VALUES(default);

-- Insert data again.
gaussdb=# INSERT INTO bigserial_type_tab VALUES(default);

-- View data.
gaussdb=# SELECT * FROM bigserial_type_tab;
 c
---
 1
 2
(2 rows)

-- Create a table.
gaussdb=# CREATE TABLE largeserial_type_tab(c LARGESERIAL);

-- Insert data.
gaussdb=# INSERT INTO largeserial_type_tab VALUES(default);

-- Insert data again.
gaussdb=# INSERT INTO largeserial_type_tab VALUES(default);

-- View data.
gaussdb=# SELECT * FROM largeserial_type_tab;
 c
---
 1
 2
(2 rows)

-- Drop the table.
gaussdb=# DROP TABLE smallserial_type_tab;
```



```
gaussdb=# DROP TABLE serial_type_tab;

gaussdb=# DROP TABLE bigserial_type_tab;
```

NOTE

SMALLSERIAL, SERIAL, BIGSERIAL, and LARGESERIAL are not real types. They are concepts used for setting a unique identifier for a table. Therefore, an integer column is created and its default value plans to be read from a sequencer. A NOT NULL constraint is used to ensure NULL is not inserted. In most cases you would also want to attach a **UNIQUE** or **PRIMARY KEY** constraint to prevent duplicate values from being inserted unexpectedly, but this is not automatic. Finally, the sequencer belongs to the column. In this case, when the column or the table is deleted, the sequencer is also deleted. Currently, you can specify a SERIAL column when creating a table or add a SERIAL column to an ordinary table in PG-compatible mode. In addition, **SERIAL** columns cannot be created in temporary tables. Because SERIAL is not a data type, columns cannot be converted to this type.

Table 7-5 shows the floating point type.

NOTE

- The REAL(p,s), DOUBLE, DOUBLE(p,s), and FLOAT(p,s) floating-point types can be used only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**. For the behavior of the REAL, FLOAT, and DEC[(p[,s])] data types when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, see Table 7-5.
- In Table 7-5, **p** is the precision, indicating the minimum acceptable total number of integral places, and **s** indicates the decimal digit.

Table 7-5 Floating point types

Name	Description	Storage Space	Range
REAL, FLOAT4	Single precision floating points, inexact The REAL data type is mapped to the double-precision floating-point number FLOAT8 when the scenario described in **NOTE** is met. For details about the application scenario, see FLOAT8.	4 bytes.	–3.402E+38 to +3.402E+38, 6-digit decimal digits.

Name	Description	Storage Space	Range
REAL(p,s)	The value range of **p** is [1,1000], and the value range of **s** is [0,*p*]. If the precision and scale are not specified, the precision **p** is **10** and the scale **s** is **0** by default. This type maps to NUMERIC. For details about the application scenario, see NUMERIC.	The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data.	Maximum 131,072 digits before the decimal point and 16,383 digits after the decimal point when the precision and scale are specified to the maximum.
DOUBLE PRECISION, FLOAT8	Double precision floating points, inexact	8 bytes.	–1.79E+308 to +1.79E+308, 15-bit decimal digits.
DOUBLE	Double precision floating points, inexact This type maps to double-precision floating-point number FLOAT8. For details about the application scenario, see FLOAT8.	8 bytes.	–1.79E+308 to +1.79E+308, 15-bit decimal digits.

Name	Description	Storage Space	Range
DOUBLE(p, s)	The value range of **p** is [1,1000], and the value range of **s** is [0,*p*]. If the precision and scale are not specified, the precision **p** is **10** and the scale **s** is **0** by default. This type maps to NUMERIC. For details about the application scenario, see NUMERIC.	The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data.	Maximum 131,072 digits before the decimal point and 16,383 digits after the decimal point when the precision and scale are specified to the maximum.
FLOAT[(p)]	Floating points, inexact The value range of precision (p) is [1,53]. The type is selected based on the precision **p**. When **p** is less than or equal to 24, the mapping type is REAL. When **p** is greater than 24 or not specified, the mapping type is DOUBLE PRECISION. In the scenario described in **NOTE**, if the precision **p** is less than or equal to 24, or it is not specified, the mapping type is FLOAT4. If **p** is greater than 24, the mapping type is DOUBLE PRECISION.	4 bytes or 8 bytes.	-

Name	Description	Storage Space	Range
FLOAT(p,s)	The value range of **p** is [1,1000], and the value range of **s** is [0,*p*]. If the precision and scale are not specified, the precision **p** is **10** and the scale **s** is **0** by default. This type maps to NUMERIC. For details about the application scenario, see NUMERIC.	The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data.	Maximum 131,072 digits before the decimal point and 16,383 digits after the decimal point when the precision and scale are specified to the maximum.
BINARY_DOUBLE	Alias for DOUBLE PRECISION, compatible with Oracle.	8 bytes.	–1.79E+308 to +1.79E+308, 15-bit decimal digits.
DEC[(p[,s])]	The value range of **p** (precision) is [1,1000], and the value range of **s** (scale) is [0,*p*]. If the scenario in **NOTE** is met and the precision and scale are not specified, the precision **p** is **10** and the scale **s** is **0** by default. This type maps to NUMERIC. For details about the application scenario, see NUMERIC.	The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data.	Maximum 131,072 digits before the decimal point and 16,383 digits after the decimal point when the precision and scale are specified to the maximum.

Name	Description	Storage Space	Range
INTEGER[(p[,s])]	The value range of **p** (precision) is [1,1000], and the value range of **s** (scale) is [0,*p*]. If the precision and scale are not specified, the precision **p** is **10** and the scale **s** is **0** by default. If the precision and scale are not specified, this type is mapped to INTEGER. If the precision and scale are specified, this type is mapped to NUMERIC.	The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data.	Maximum 131,072 digits before the decimal point and 16,383 digits after the decimal point when the precision and scale are specified to the maximum. If the precision and scale are not specified, the value ranges from –2,147,483,648 to +2,147,483,647.

NOTE

When **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, the input and output ranges of the FLOAT[(p[,s])], DOUBLE[(p[,s])], DEC[(p[,s])] and REAL[(p[,s])] types are different.

- Input format: The value can be a number or a numeric string.
  - If the value of **SQL_MODE** contains **strict_trans_tables**, an error is reported if the input value is invalid or exceeds the range.
  - If the value of **SQL_MODE** does not contain **strict_trans_tables** and the input value is invalid or exceeds the range, a warning message is reported and **0** or the truncated value is returned.
- Output format: The output format of the DEC[(p[,s])] type is different from that of the mapping NUMERIC[(p[,s])] type when the precision and scale are not specified. If NUMERIC has no precision or scale, the decimal part is retained during type conversion. For details, see the output format in the scenario where no parameter is set. If DEC has no precision or scale, the precision is **10** and the scale is **0** during type conversion.

Example:

```
-- Create a table.
gaussdb=# CREATE TABLE float_type_t2
(
    FT_COL1 INTEGER,
    FT_COL2 FLOAT4,
    FT_COL3 FLOAT8,
    FT_COL4 FLOAT(3),
    FT_COL5 BINARY_DOUBLE,
    FT_COL6 DECIMAL(10,4),
    FT_COL7 INTEGER(6,3)
);
```



```
-- Insert data.
gaussdb=# INSERT INTO float_type_t2 VALUES(10,10.365456,123456.1234,10.3214, 321.321, 123.123654,
123.123654);

-- View data.
gaussdb=# SELECT * FROM float_type_t2 ;
 ft_col1 | ft_col2 |   ft_col3   | ft_col4 | ft_col5 | ft_col6  | ft_col7
---------+---------+-------------+---------+---------+----------+---------
      10 | 10.3655 | 123456.1234 | 10.3214 | 321.321 | 123.1237 | 123.124
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE float_type_t2;

-- Examples: REAL(p,s), FLOAT(p,s), DOUBLE, and DOUBLE(p,s).
gaussdb=# CREATE DATABASE gaussdb_m  WITH dbcompatibility  'B';
gaussdb=# \c gaussdb_m
-- Set compatible version control parameters.
gaussdb_m=# SET b_format_version='5.7';
gaussdb_m=# SET b_format_dev_version='s1';
-- Create a table.
gaussdb_m=# CREATE TABLE t1(a real(10,2), b float(10,2), c double, d double(10,2));
-- Insert data.
gaussdb_m=# INSERT INTO t1 VALUES (1000.12, 2000.23, 3000.34, 4000.45);
-- Query data in the table.
gaussdb_m=# SELECT * FROM t1;
   a    |    b    |    c    |    d
--------+---------+---------+---------
 1000.12 | 2000.23 | 3000.34 | 4000.45
(1 row)

-- Delete the table and database.
gaussdb_m=# DROP TABLE t1;
gaussdb_m=# \c postgres;
gaussdb=# DROP DATABASE gaussdb_m;

-- Reset parameters.
gaussdb=# RESET ALL;
```

7.3.2 Monetary Types

The monetary type stores a currency amount with fixed fractional precision.

The range shown in **Table 7-6** assumes there are two fractional digits. Input is accepted in a variety of formats, including integer and floating-point literals, as well as typical currency formatting, such as "$1,000.00". Output is generally in the last format but depends on the locale.

Table 7-6 Monetary type

Name	Description	Storage Space	Range
money	Currency amount	8 bytes	-92233720368547758.08 ~ +92233720368547758.07

Values of the numeric, int, and bigint data types can be cast to money. Conversion from the real or double precision data type to a money value can be done by casting to numeric type first, for example:


```
gaussdb=# SELECT '12.34'::float8::numeric::money;
 money
--------
 $12.34
(1 row)
```

However, this is not recommended. Floating point numbers should not be used to handle money due to the potential for rounding errors.

A money value can be cast to numeric without loss of precision. Conversion to other types could potentially lose precision, and must also be done in two stages:

```
gaussdb=# SELECT '52093.89'::money::numeric::float8;
  float8
----------
 52093.89
(1 row)
```

When a money value is divided by another money value, the result is of the double precision type (that is, a pure number, not money); the currency units cancel each other out in the division.

7.3.3 Boolean Types

Table 7-7 lists the Boolean types supported by GaussDB.

Table 7-7 Boolean Types

Name	Description	Storage Space	Value
BOOLEAN	Boolean	1 byte	• **true** • **false** • **null**: unknown

- Valid literal values for the "true" state are:
  TRUE, **'t'**, **'true'**, **'y'**, **'yes'**, **'1'**, **'TRUE'**, **true**, **on**, and all non-zero values.
- Valid literal values for the "false" state include:
  FALSE, **'f'**, **'false'**, **'n'**, **'no'**, **'0'**, **0**, **'FALSE'**, **false**, and **off**.

TRUE and **FALSE** are standard expressions, compatible with SQL statements.

Examples

Boolean values are displayed using the letters t and f.

```
-- Create a table.
gaussdb=# CREATE TABLE bool_type_t1
(
    BT_COL1 BOOLEAN,
    BT_COL2 TEXT
);

-- Insert data.
gaussdb=# INSERT INTO bool_type_t1 VALUES (TRUE, 'sic est');

gaussdb=# INSERT INTO bool_type_t1 VALUES (FALSE, 'non est');
```



```
-- View data.
gaussdb=# SELECT * FROM bool_type_t1;
 bt_col1 | bt_col2
---------+---------
 t       | sic est
 f       | non est
(2 rows)

gaussdb=# SELECT * FROM bool_type_t1 WHERE bt_col1 = 't';
 bt_col1 | bt_col2
---------+---------
 t       | sic est
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE bool_type_t1;
```

7.3.4 Character Types

Table 7-8 lists the character types supported by GaussDB. For string operators and related built-in functions, see **Character Processing Functions and Operators**.

Table 7-8 Character types

Name	Description	Storage Space
CHAR(n) CHARACTER(n) NCHAR(n)	Fixed-length string. Empty characters are filled in with blank spaces. **n** indicates the string length. If it is not specified, the default precision **1** is used.	The maximum size is 10 MB.

Name	Description	Storage Space
	When **sql_compatibility** is set to **'B'**, if **b_format_version** is set to **'5.7'** and **b_format_dev_version** is set to **'s1'**: ● Type: **n** indicates the string length. The value range is [0,10485760]. If the precision **n** is not specified, the default precision is 1. Fixed-length string. Empty characters are filled in with blank spaces. ● Input – Data type of table columns and temporary variables: If the number of characters in an input string is within the range, the string can be entered normally. If the value of **sql_mode** contains **strict_trans_tables**, an error is reported. Otherwise, the string is truncated based on the maximum string length specified by **n** and an alarm is generated. ● Output – Data type of table columns and temporary variables: If the value of **sql_mode** contains **pad_char_to_full_length**, the character string containing spaces at the end is output. Otherwise, the character string without spaces at the end is output. – Function parameter and return value or stored procedure parameter: The length cannot be verified. For example, if the input parameter of a user-defined function is of the CHAR(5) type and a string **'123456'** is input, the parameter can be directly transferred without length verification.	The value contains a maximum of 10M characters.

Name	Description	Storage Space
VARCHAR(n) CHARACTER VARYING(n)	Variable-length string. In PostgreSQL-compatible mode, **n** indicates the string length. In other compatibility modes, **n** indicates the byte length.	The maximum value of **n** is **10485760** (10 MB). If **n** is not specified, the maximum storage length is 1 GB – 85 bytes – Length of the first *n* columns. For example, the maximum length of (a int, b varchar) is 1,073,741,735 bytes (= 1 GB – 85 bytes – 4 bytes).
	When **sql_compatibility** is set to **'B'**, if **b_format_version** is set to **'5.7'** and **b_format_dev_version** is set to **'s1'**: ● **n** indicates the byte length. The value range is [0,10485760]. If the precision **n** is not specified, there is no length limit by default. The length is the same as that of the TEXT type. ● Input – Data type of table columns and temporary variables: If the number of characters in an input string is within the range, the string can be entered normally. If the value of **sql_mode** contains **strict_trans_tables**, an error is reported. Otherwise, the string is truncated based on the maximum string length specified by **n** and an alarm is generated. ● Output: The original character string is output.	The maximum value of **n** is **10485760** (10 MB). If **n** is not specified, the maximum storage length is 1 GB – 85 bytes – Length of the first *n* columns. For example, the maximum length of (a int, b varchar) is 1,073,741,735 bytes (= 1 GB – 85 bytes – 4 bytes).
VARCHAR2(n)	Variable-length string. It is an alias for VARCHAR(n) type, which is compatible with Oracle.	The maximum value of **n** is **10485760** (10 MB). If **n** is not specified, the maximum storage length is 1 GB – 85 bytes – Length of the first *n* columns. For example, the maximum length of (a int, b varchar) is 1,073,741,735 bytes (= 1 GB – 85 bytes – 4 bytes).

Name	Description	Storage Space
NVARCHAR2(n)	Variable-length string. In the SQL_ASCII character set, **n** indicates bytes. In the non-SQL_ASCII character set, **n** indicates characters.	The maximum value of **n** is **10485760** (10 MB). If **n** is not specified, the maximum storage length is 1 GB – 85 bytes – Length of the first *n* columns. For example, the maximum length of (a int, b varchar) is 1,073,741,735 bytes (= 1 GB – 85 bytes – 4 bytes).
TEXT	Variable-length string.	The maximum size is 1 GB minus 1 byte. However, the size of the column description header and the size of the tuple (less than 1 GB minus 1 byte) where the column is located must also be considered. Therefore, the maximum size of the TEXT type may be less than 1 GB minus 1 byte.

Name	Description	Storage Space
CLOB	Large text object, which is compatible with Oracle.	For Astore, the maximum size is 32 TB minus 1 byte. However, the size of the column description header and the size of the tuple (less than 32 TB minus 1 byte) where the column is located must also be considered. Therefore, the maximum size of the CLOB type may be less than 32 TB minus 1 byte. For Ustore, the maximum size is 1 GB minus 1 byte. However, the size of the column description header and the size of the tuple (less than 1 GB minus 1 byte) where the column is located must also be considered. Therefore, the maximum size of the CLOB type may be less than 1 GB minus 1 byte. For Ustore, the maximum size is 1 GB minus 1 byte. However, the size of the column description header and the size of the tuple (less than 1 GB minus 1 byte) where the column is located must also be considered. Therefore, the maximum size of the CLOB type may be less than 1 GB minus 1 byte.

Name	Description	Storage Space
TINYTEXT MEDIUMTEXT LONGTEXT	If **sql_compatibility** is set to **'B'**, this parameter takes effect only when **b_format_version** is set to **'5.7'** and **b_format_dev_version** is set to **'s1'**. Converts a data type to the TEXT type. The application scenario is the same as that of the TEXT type.	The maximum size is 1 GB minus 1 byte. However, the size of the column description header and the size of the tuple (less than 1 GB minus 1 byte) where the column is located must also be considered. Therefore, the maximum size of this type may be less than 1 GB minus 1 byte.

NOTE

1. In addition to the restriction on the size of each column, the total size of each tuple cannot exceed 1 GB minus 1 byte and is affected by the control header information of the column, the control header information of the tuple, and whether null fields exist in the tuple.
2. NCHAR is the alias of the bpchar type, and VARCHAR2(n) is the alias of the VARCHAR(n) type.
3. Only advanced package dbe_lob supports CLOBs whose size is greater than 1 GB. System functions do not support CLOBs whose size is greater than 1 GB.
4. In A-compatible mode, the received empty string is converted to null by default.

In GaussDB, there are two other fixed-length character types, as described in **Table 7-9**. The name type exists only for the storage of identifiers in the internal system catalogs and is not intended for use by general users. Its length is currently defined as 64 bytes (63 usable characters plus terminator). The char type uses only one byte of storage. It is internally used in the system catalogs as a simplistic enumeration type.

Table 7-9 Special character types

Name	Description	Storage Space
name	Internal type for object names	64 bytes
"char"	Single-byte internal type	1 byte

7.3.5 Binary Types

Table 7-10 lists the binary types supported by GaussDB.

Table 7-10 Binary types

Name	Description	Storage Space
BLOB	Binary large object (BLOB). Currently, BLOB only supports the following external access APIs: • DBE_LOB.GET_LENGTH • DBE_LOB.READ • DBE_LOB.WRITE • DBE_LOB.WRITE_APPEND • DBE_LOB.COPY • DBE_LOB.ERASE For details about the APIs, see **DBE_LOB**.	For Ustore, the maximum size is 1 GB minus 1 byte. However, the size of the column description header and the size of the tuple (less than 1 GB minus 1 byte) where the column is located must also be considered. Therefore, the maximum size of the BLOB type may be less than 1 GB minus 1 byte.
	When **sql_compatibility** is set to **'B'**, if **b_format_version** is set to **'5.7'** and **b_format_dev_version** is set to **'s1'**, the BLOB type is mapped to the BYTEA type, and the alias is BYTEA.	For details about the storage specifications, see the BYTEA type.
TINYBLOB MEDIUMBLOB LONGBLOB	Binary large object (BLOB).	For details about the storage specifications, see the BYTEA type.
	This type can be used only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**. The type is mapped to the BYTEA type, and the alias is BYTEA.	

Name	Description	Storage Space
RAW	Variable-length hexadecimal string.	4 bytes plus the actual binary string. The maximum size is 1 GB minus 1 byte. However, the size of the column description header and the size of the tuple (less than 1 GB minus 1 byte) where the column is located must also be considered. Therefore, the maximum size of this type may be less than 1 GB minus 1 byte.
BYTEA	Variable-length binary string.	4 bytes plus the actual binary string. The maximum size is 1 GB minus 1 byte. However, the size of the column description header and the size of the tuple (less than 1 GB minus 1 byte) where the column is located must also be considered. Therefore, the maximum size of this type may be less than 1 GB minus 1 byte.
BYTEAWITHOUTORDERWITHEQUALCOL	Variable-length binary character string (new type for the encryption feature. If the encryption type of an encrypted column is specified as deterministic encryption, the column type is BYTEAWITHOUTORDER-WITHEQUALCOL). The original data type is displayed when an encrypted table is printed by running the meta command.	4 bytes plus the actual binary string. The maximum value is 1,073,741,771 bytes (1 GB minus 53 bytes).

Name	Description	Storage Space
BYTEAWITHOUTORDERCOL	Variable-length binary character string (new type for the encryption feature. If the encryption type of an encrypted column is specified as random encryption, the column type is BYTEAWITHOUTORDER-COL). The original data type is displayed when an encrypted table is printed by running the meta command.	4 bytes plus the actual binary string. The maximum value is 1,073,741,771 bytes (1 GB minus 53 bytes).
_BYTEAWITHOUTORDERWITHEQUALCOL	Variable-length binary string, which is a new type for the encryption feature.	4 bytes plus the actual binary string. The maximum value is 1,073,741,771 bytes (1 GB minus 53 bytes).
_BYTEAWITHOUTORDERCOL	Variable-length binary string, which is a new type for the encryption feature.	4 bytes plus the actual binary string. The maximum value is 1,073,741,771 bytes (1 GB minus 53 bytes).

NOTE

- In addition to the size limit of each column, the total size of each tuple cannot exceed 1 GB minus 1 byte.
- BYTEAWITHOUTORDERWITHEQUALCOL, BYTEAWITHOUTORDERCOL, _BYTEAWITHOUTORDERWITHEQUALCOL, and _BYTEAWITHOUTORDERCOL cannot be directly used to create a table.
- RAW(*n*), where *n* indicates the recommended byte length and is not used to verify the byte length of the input raw type.
- When **sql_compatibility** is set to **'B'**, if **b_format_version** is set to **'5.7'** and **b_format_dev_version** is set to **'s1'**, the TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB types are displayed as BYTEA. For example, when the table structure is queried, the TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB types are displayed as BYTEA.

Example:

```
-- Create a table.
gaussdb=# CREATE TABLE blob_type_t1
(
    BT_COL1 INTEGER,
    BT_COL2 BLOB,
    BT_COL3 RAW,
    BT_COL4 BYTEA
) ;

-- Insert data.
gaussdb=# INSERT INTO blob_type_t1 VALUES(10,empty_blob(),
HEXTORAW('DEADBEEF'),E'\\xDEADBEEF');
```



```
-- Query data in the table.
gaussdb=# SELECT * FROM blob_type_t1;
 bt_col1 | bt_col2 | bt_col3  |  bt_col4
---------+---------+----------+------------
      10 |         | DEADBEEF | \xdeadbeef
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE blob_type_t1;

-- Example: TINYBLOB, MEDIUMBLOB, and LONGBLOB types
gaussdb=# CREATE DATABASE gaussdb_m WITH dbcompatibility  'B';
gaussdb=# \c gaussdb_m
-- Set compatible version control parameters.
gaussdb_m=# SET b_format_version='5.7';
gaussdb_m=# SET b_format_dev_version='s1';
-- Create a table.
gaussdb_m=# CREATE TABLE t1(a tinyblob, b blob, m mediumblob, l longblob);
-- Insert data.
gaussdb_m=# INSERT INTO t1 VALUES ('tinyblobtest', 'blobtest', 'mediumblobtest', 'longblobtest');
-- Query data in the table.
gaussdb_m=# SELECT * FROM t1;
      a       |    b     |       m        |     l
--------------+----------+----------------+--------------
 tinyblobtest | blobtest | mediumblobtest | longblobtest
(1 row)

-- Drop the table and database.
gaussdb_m=# DROP TABLE t1;
gaussdb_m=# \c postgres;
gaussdb=# DROP DATABASE gaussdb_m;
-- Reset parameters.
gaussdb=# \c RESET ALL;
```

7.3.6 Date/Time Types

Table 7-11 lists the date/time types that can be used in GaussDB. For the operators and built-in functions of the types, see **Date and Time Processing Functions and Operators**.

NOTE

If the time format of another database is different from that of GaussDB, modify the value of the **DateStyle** parameter to keep them consistent.

Table 7-11 Date/Time types

Name	Description	Storage Space
DATE	Date. Minimum value: **4713-01-01BC** (4713 B.C.). Maximum value: **5874897-12-31AD** (5874897 A.C.) After **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, the input and output formats and ranges are different. ● Input format: YYYY-MM-DD with separators and YYYYMMDD without separators are supported. – In the scenario where separators are used, the plus sign (+) and colon (:) cannot be used as separators between the year, month, and day of a date. (Some separators are not supported when they are used together. The specifications inherit the existing implementation. Example: **date'2020-12?12'**.) The input format is not affected by the **DateStyle** parameter. – In the scenario where separators are not used, the output result may be incorrect though no error is reported. ● Output format: Only YYYY-MM-DD is supported and the format is not affected by the **DateStyle** parameter. ● Value range: 4713-01-01 BC to 5874897-12-31 AD. – If the value of **SQL_MODE** contains **strict_trans_tables**, an error is reported if the value is invalid or exceeds the range. – If the value of **SQL_MODE** does not contain	4 bytes (8 bytes in A compatibility schema)

Name	Description	Storage Space
	strict_trans_tables, the values of year, month, and day can be **0**. However, the values will be converted to valid values in the sequence of year, month, and day. For example, **'0000-00-10'** will be converted to **0002-12-10 BC**. If the input is invalid or exceeds the range, a warning message is reported and the value **0000-00-00 00:00:00** is returned. **NOTE** For A compatibility, the database treats empty strings as **NULL** and replaces DATE with TIMESTAMP(0) WITHOUT TIME ZONE.	

Name	Description	Storage Space
TIME [(p)] [WITHOUT TIME ZONE]	Time within one day (without a time zone). **p** indicates the precision after the decimal point. The value ranges from **0** to **6**. Minimum value: **00:00:00**. Maximum value: **24:00:00**. After **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, they are not limited to the time in a day. For example, they can indicate concepts such as duration and relative time, and the format, range, and precision are different. ● Input format: [D] hh:mm:ss.ffffff with separators and hhmmss.ffffff without separators are supported. **D** indicates the number of days. The value is an integer (negative numbers are supported). This parameter is optional. The value is converted into the number of hours on the basis of 24 hours in a day. The value is displayed after hh is added. Only colons (:) can be used as separators. ● Output format: hh:mm:ss.ffffff. All zeros at the end of the decimal part are automatically ignored during display. ● Value range: –838:59:59.000000 to 838:59:59.000000. – If the value of **SQL_MODE** contains **strict_trans_tables**, **00:00:00** is returned for invalid input. If the value exceeds the range, an error is reported. – If the value of **SQL_MODE** does not contain **strict_trans_tables**, **00:00:00** is returned for invalid input. If the input is valid but	8 bytes

Name	Description	Storage Space
	exceeds the range, the nearest boundary value is returned. For example, inputting 838:59:59.000001 returns 838:59:59, and inputting –838:59:59.000001 returns –838:59:59. • Precision: **p** indicates the precision after the decimal point. The value ranges from 0 to 6. If **p** is set to a value greater than 6, the value **6** is used. – When it is used as the data type of a table column, the default precision is **0**. – When it is used as an expression (for example, time '10:10:10.123456'), the default precision is **6**.	
TIME [(p)] [WITH TIME ZONE]	Time within one day (with a time zone). **p** indicates the precision after the decimal point. The value ranges from **0** to **6**. Minimum value: **00:00:00+15:59**. Maximum value: **24:00:00–15:59**.	12 bytes

Name	Description	Storage Space
TIMESTAMP[(p)] [WITHOUT TIME ZONE]	Date and time (without a time zone). **p** indicates the precision after the decimal point. The value ranges from **0** to **6**. Minimum value: **4713-11-24 BC 00:00:00.000000** (4713 B.C.). Maximum value: **294277-01-09 AD 00:00:00.000000** (294277 A.D.). After **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, they will be replaced with the TIMESTAMP[(p)] [WITH TIME ZONE] type. However, the format, range, and precision specifications are different. • Input format: YYYY-MM-DD hh:mm:ss.ffffff+timezone with separators and YYYYMMDDhhmmss.ffffff without separators are supported. – If separators are used, the plus sign (+) and colon (:) cannot be used as separators between the year, month, and day. (Some separators are not supported when they are used together. The specifications inherit the existing implementation. For example, timestamp '2020-12?12 00:00:00'.) Only colons (:) can be used as separators in the time part. The input format is not affected by the **DateStyle** parameter. – In the scenario where separators are not used and the input is incomplete, the output result may be incorrect though no error is reported. – If the year has only two digits, 00–69 corresponds to	8 bytes

Name	Description	Storage Space
	2000–2069, and 70–99 corresponds to 1970–1999. – If the year contains only one digit, for example, timestamp '1-1-1 00:00:00', the result is '0001-01-01 00:00:00'. ● Output format: Only YYYY-MM-DD hh:mm:ss.ffffff is supported and the format is not affected by the **DateStyle** parameter. All zeros at the end of the decimal part of the time are automatically ignored. ● Range – If the value of **SQL_MODE** contains **strict_trans_tables**, the value range is the same as that of TIMESTAMP[(p)] WITH TIME ZONE. If the value is invalid or exceeds the range, an error is reported. – If the value of **SQL_MODE** does not contain **strict_trans_tables**, the values of year, month, and day can be **0**. However, the values will be converted to valid values in the sequence of year, month, and day. For example, **'0000-00-10 00:00:00'** will be converted to **0002-12-10 00:00:00 BC**. If the input is invalid or exceeds the range, a warning message is reported and the value **0000-00-00 00:00:00** is returned. ● Precision: **p** indicates the precision after the decimal point. The value ranges from 0 to 6. If **p** is set to a value greater than 6, the value **6** is used. – When it is used as the data type of a table column, the default precision is **0**. – When it is used as an expression (for example,	

Name	Description	Storage Space
	timestamp '2000-01-01 00:00:00.123456'), the default precision is **6**. • Time zone: The value ranges from –15:59 to +15:59. You can also use the time zone name (see the system view **PG_TIMEZONE_NAMES**). The time zone specifications are the same as those of the TIMESTAMP[(p)] WITH TIME ZONE type. Converts the time value of a specified time zone to the time of the UTC-0 time zone. When it is displayed, the time in the UTC-0 time zone is converted to the time in the current time zone of the server. You can run **SHOW TIME ZONE** to view the time zone of the server. Therefore, when you use the SET TIME ZONE statement to change the time zone of the server, the displayed result also changes.	

Name	Description	Storage Space
TIMESTAMP[(p)] [WITH TIME ZONE]	Date and time (with a time zone). TIMESTAMP is also called TIMESTAMPTZ. **p** indicates the precision after the decimal point. The value ranges from **0** to **6**. Minimum value: **4713-11-24 BC 00:00:00.000000** (4713 B.C.). Maximum value: **294277-01-09 AD 00:00:00.000000** (294277 A.D.). Time zone update: In some countries or regions, the time zone information is frequently updated due to political, economic, and war factors. Therefore, the database system needs to modify the time zone file to ensure that the time is correct. Currently, the GaussDB time zone type involves only timestamp with timezone. When a new time zone file takes effect, the existing data is not changed, and the new data is adjusted based on the time zone file information.	8 bytes
SMALLDATETIME	Date and time (without a time zone). The precision level is minute. 30s to 59s are rounded into one minute. Minimum value: **4713-11-24BC 00:00:00.000000** (4713 B.C.). Maximum value: **294277-01-09AD 00:00:00.000000** (294277 A.D.).	8 bytes
INTERVAL DAY (l) TO SECOND (p)	Specifies the time interval (*X* days *X* hours *X* minutes *X* seconds). • **l**: indicates the precision of days. The value ranges from 0 to 6. For compatibility, the precision functions are not supported. • **p**: indicates the precision of seconds. The value ranges from 0 to 6. The digit 0 at the end of a decimal number is not displayed.	16 bytes

Name	Description	Storage Space
INTERVAL [FIELDS] [(p)]	Time interval. • **FIELDS**: **YEAR**, **MONTH**, **DAY**, **HOUR**, **MINUTE**, **SECOND**, **DAY TO HOUR**, **DAY TO MINUTE**, **DAY TO SECOND**, **HOUR TO MINUTE**, **HOUR TO SECOND**, or **MINUTE TO SECOND**. • **p**: indicates the precision of seconds. The value ranges from 0 to 6. **p** takes effect only when **FIELDS** is set to **SECOND**, **DAY TO SECOND**, **HOUR TO SECOND**, or **MINUTE TO SECOND**. The digit 0 at the end of a decimal number is not displayed.	12 bytes
reltime	Relative time interval. • The format is as follows: X years X mons X days XX:XX:XX. • The Julian calendar is used. It specifies that a year has 365.25 days and a month has 30 days. The relative time interval needs to be calculated based on the input value.	4 bytes
abstime	Date and time. • Format: YYYY-MM-DD hh:mm:ss +timezone. • The value range is from 1901-12-13 20:45:53 GMT to 2038-01-18 23:59:59 GMT. The precision is second.	4 bytes

Name	Description	Storage Space
datetime[(p)]	Date and time. This parameter takes effect only when **sql_compatibility** is set to **'B**', **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**. The datetime[(p)] type is replaced by the TIMESTAMP[(p)] WITHOUT TIME ZONE type, but the format, range, precision, and time zone processing specifications are different. ● Input format: YYYY-MM-DD hh:mm:ss.ffffff+timezone with separators and YYYYMMDDhhmmss.ffffff without separators are supported. – If separators are used, the plus sign (+) and colon (:) cannot be used as separators between the year, month, and day. (Some separators are not supported when they are used together. The specifications inherit the existing implementation. For example, **datetime '2020-12?12 00:00:00'**.) Only colons (:) can be used as separators in the time part. The input format is not affected by the **DateStyle** parameter. – In the scenario where separators are not used and the input is incomplete, the output result may be incorrect though no error is reported. – If the year has only two digits, 00–69 corresponds to 2000–2069, and 70–99 corresponds to 1970–1999. – If the year contains only one digit, for example, datetime '1-1-1 00:00:00', the result is '0001-01-01 00:00:00'.	8 bytes

Name	Description	Storage Space
	• Output format: Only YYYY-MM-DD hh:mm:ss.ffffff is supported and the format is not affected by the **DateStyle** parameter. All zeros at the end of the decimal part of the time are automatically ignored. • Range – If the value of **SQL_MODE** contains **strict_trans_tables**, the value range is the same as that of TIMESTAMP[(p)] WITHOUT TIME ZONE. If the value is invalid or exceeds the range, an error is reported. – If the value of **SQL_MODE** does not contain **strict_trans_tables**, the values of year, month, and day can be 0. However, the values will be converted to valid values in the sequence of year, month, and day. For example, **'0000-00-10 00:00:00'** will be converted to **0002-12-10 00:00:00 BC**. If the input is invalid or exceeds the range, a warning message is reported and the value **0000-00-00 00:00:00** is returned. • Precision: **p** indicates the precision after the decimal point. The value ranges from 0 to 6. If the specified precision exceeds 6, the value **6** is used. – When it is used as the data type of a table column, the default precision is **0**. – When it is used as an expression (for example, datetime '2000-01-01 00:00:00.123456'), the default precision is **6**. • Time zone: The value ranges from –15:59 to +15:59. You can also use the time zone name (see the system view	

Name	Description	Storage Space
	PG_TIMEZONE_NAMES). The time zone specifications are the same as those of the TIMESTAMP[(p)] WITH TIME ZONE type. It converts the time value of a specified time zone to the time of the current time zone of the server. You can run **SHOW TIME ZONE** to view the time zone of the server. If the converted date and time are saved to the table, the time zone is not converted when the time zone of the server changes.	

Name	Description	Storage Space
year[(w)]	Year. It takes effect only after **sql_compatibility** is set to **'B'**. ● Input formats: – A string of four digits, ranging from '1901' to '2155' (that is, the years from 1901 to 2155). – Four digits, ranging from 1901 to 2155 (that is, the years from 1901 to 2155). – A string of one or two digits, ranging from '0' to '99'. The values '0' to '69' indicate the years from 2000 to 2069, and the values '70' to '99' indicate the years from 1970 to 1999. **'0'** and **'00'** indicate the year 2000. – One or two digits, ranging from **0** to **99**. The values **1** to **69** indicate the years from 2001 to 2069, and **70** to **99** indicate the years from 1970 to 1999. The value **0** indicates 0000. – Return values of other time functions, for example, now(). ● Output format: Only the YYYY format is supported. **w** indicates the number of digits in the output format. Only four digits are supported. If not specified, the default value is **4**. ● Value range: **1901** to **2155**. – If the value of **SQL_MODE** contains **strict_trans_tables**, an error is reported if the value is invalid or exceeds the range. – If the value of **SQL_MODE** does not contain **strict_trans_tables**, the invalid value is converted to **0000** and inserted, and an alarm is generated.	1 byte.

NOTE

1. The data of the time type automatically ignores all zeros at the end of the data when it is displayed.
2. The default value of **p** is **6**.
3. For the INTERVAL type, the date and time are stored in the int32 and double types in the system. Therefore, the value ranges of the two types are the same as those of the corresponding data type.
4. If the insertion time is out of the range, the system may not report an error, but may not ensure that the operation is normal.

NOTE

If the values of **a_format_version** and **a_format_dev_version** are **10c** and **s1**, the default DATE value is determined by the following:

- Year: returned through SYSDATE
- Month: returned through SYSDATE
- Day: 01 (first day of the month)
- Hour, minute, and second: all 0

Example:

```
-- Create a table.
gaussdb=# CREATE TABLE date_type_tab(coll date);

-- Insert data.
gaussdb=# INSERT INTO date_type_tab VALUES (date '12-10-2010');

-- View data.
gaussdb=# SELECT * FROM date_type_tab;
       coll
---------------------
 2010-12-10(1 row)

-- Drop the table.
gaussdb=# DROP TABLE date_type_tab;

-- Create a table.
gaussdb=# CREATE TABLE time_type_tab (da time without time zone ,dai time with time zone,dfgh
timestamp without time zone,dfga timestamp with time zone, vbg smalldatetime);

-- Insert data.
gaussdb=# INSERT INTO time_type_tab VALUES ('21:21:21','21:21:21 pst','2010-12-12','2013-12-11
pst','2003-04-12 04:05:06');

-- View data.
gaussdb=# SELECT * FROM time_type_tab;
    da    |     dai     |        dfgh         |          dfga          |        vbg
----------+-------------+---------------------+------------------------+---------------------
 21:21:21 | 21:21:21-08 | 2010-12-12 00:00:00 | 2013-12-11 16:00:00+08 | 2003-04-12 04:05:00
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE time_type_tab;

-- Create a table.
gaussdb=# CREATE TABLE day_type_tab (a int,b INTERVAL DAY(3) TO SECOND (4));

-- Insert data.
gaussdb=# INSERT INTO day_type_tab VALUES (1, INTERVAL '3' DAY);
```



```
-- View data.
gaussdb=# SELECT * FROM day_type_tab;
 a |   b
---+--------
 1 | 3 days
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE day_type_tab;

-- Create a table.
gaussdb=# CREATE TABLE year_type_tab(a int, b interval year (6));

-- Insert data.
gaussdb=# INSERT INTO year_type_tab VALUES(1,interval '2' year);

-- View data.
gaussdb=# SELECT * FROM year_type_tab;
 a |    b
---+---------
 1 | 2 years
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE year_type_tab;

-- Example of the datetime and timestamp data types in B-compatible mode.
-- Create a B-compatible database.
-- Switch to the B-compatible database.
gaussdb=# CREATE DATABASE gaussdb_m dbcompatibility='B';
gaussdb=# \c gaussdb_m;

-- Set compatible version control parameters.
gaussdb_m=# SET b_format_version = '5.7';
gaussdb_m=# SET b_format_dev_version = 's1';

-- Create a table.
gaussdb_m=# CREATE TABLE datetime_typ_tab(col1 datetime, col2 timestamp);

-- Insert data.
gaussdb_m=# INSERT INTO datetime_typ_tab VALUES ('2003-04-12 04:05:06+09:00', '2003-04-12
04:05:06+09:00');

-- View the data.
gaussdb_m=# SELECT * FROM datetime_typ_tab;
        col1         |        col2
---------------------+---------------------
 2003-04-12 03:05:06 | 2003-04-12 03:05:06
(1 row)

-- Drop the table.
gaussdb_m=# DROP TABLE datetime_typ_tab;

-- Create a table.
gaussdb_m=# CREATE TABLE year_typ_tab(col1 year, col2 year(4));

-- Insert data.
gaussdb_m=# INSERT INTO year_typ_tab VALUES ('2023', now());

-- View the data.
gaussdb_m=# SELECT * FROM year_typ_tab;
 col1 | col2
------+------
 2023 | 2023
(1 row)

-- Delete the table and database.
gaussdb_m=# DROP TABLE year_typ_tab;
gaussdb_m=# \c postgres;
```



```
gaussdb=# DROP DATABASE gaussdb_m;

-- Reset parameters.
gaussdb=# RESET ALL;
```

Date Inputs

Date and time input is accepted in almost any reasonable formats, including ISO 8601 and SQL-compatible. The system allows you to customize the sequence of day, month, and year in the date input. Set the **DateStyle** parameter to **MDY** to select month-day-year interpretation, **DMY** to select day-month-year interpretation, or **YMD** to select year-month-day interpretation.

Remember that any date or time literal input needs to be enclosed with single quotation marks (''), and the syntax is as follows:

```
type [ ( p ) ] 'value'
```

The **p** that can be selected in the precision statement is an integer, indicating the number of fractional digits in the **seconds** column. Table 7-12 shows the input formats of the date type.

Table 7-12 Date input formats

Example	Description
1999-01-08	ISO 8601 (recommended format). January 8, 1999 in any format.
January 8, 1999	Unambiguous in any **datestyle** input mode
1/8/1999	January 8 in MDY format. August 1 in DMY format.
1/18/1999	January 18 in MDY format, rejected in other formats.
01/02/03	• January 2, 2003 in MDY format. • February 1, 2003 in DMY format. • February 3, 2001 in YMD format.
1999-Jan-08	January 8 in any mode
Jan-08-1999	January 8 in any mode
08-Jan-1999	January 8 in any mode
99-Jan-08	January 8 in **YMD** mode, else error
08-Jan-99	January 8, except error in YMD format.
Jan-08-99	January 8, except error in YMD format.
19990108	ISO 8601. January 8, 1999 in any format.
990108	ISO 8601. January 8, 1999 in any format.
1999.008	Year and day of year.
J2451187	Julian date.

Example	Description
January 8, 99 BC	Year 99 B.C.

Example:

```
-- Create a table.
gaussdb=# CREATE TABLE date_type_tab(coll date);

-- Insert data.
gaussdb=# INSERT INTO date_type_tab VALUES (date '12-10-2010');

-- View data.
gaussdb=# SELECT * FROM date_type_tab;
        coll
---------------------
 2010-12-10(1 row)

-- View the date format.
gaussdb=# SHOW datestyle;
 DateStyle
-----------
 ISO, MDY
(1 row)

-- Set the date format.
gaussdb=# SET datestyle='YMD';
SET

-- Insert data.
gaussdb=# INSERT INTO date_type_tab VALUES(date '2010-12-11');

-- View data.
gaussdb=# SELECT * FROM date_type_tab;
        coll
---------------------
 2010-12-10 2010-12-11(2 rows)

-- Drop the table.
gaussdb=# DROP TABLE date_type_tab;
```

Time

The time-of-day types are **TIME [(p)] [WITHOUT TIME ZONE]** and **TIME [(p)] [WITH TIME ZONE]**. **TIME** alone is equivalent to **TIME WITHOUT TIME ZONE**.

If a time zone is specified in the input for **TIME WITHOUT TIME ZONE**, it is silently ignored.

For details about the time input types, see **Table 7-14**. For details about time zone input types, see **Table 7-13**.

Table 7-13 Time input types

Example	Description
05:06.8	ISO 8601
4:05:06	ISO 8601

Example	Description
4:05	ISO 8601
040506	ISO 8601
4:05 AM	Same as 04:05. Input hours must be less than or equal to 12.
4:05 PM	Same as 16:05. Input hours must be less than or equal to 12.
04:05:06.789-8	ISO 8601
04:05:06-08:00	ISO 8601
04:05-08:00	ISO 8601
040506-08	ISO 8601
04:05:06 PST	Time zone specified by abbreviation
2003-04-12 04:05:06 America/New_York	Time zone specified by full name

Table 7-14 Time zone input types

Example	Description
PST	Abbreviation (for Pacific Standard Time)
America/New_York	Full time zone name
-8:00	ISO-8601 offset for PST
-800	ISO-8601 offset for PST
-8	ISO-8601 offset for PST

Example:

```
gaussdb=# SELECT time '04:05:06';
   time
----------
 04:05:06
(1 row)

gaussdb=# SELECT time '04:05:06 PST';
   time
----------
 04:05:06
(1 row)

gaussdb=# SELECT time with time zone '04:05:06 PST';
   timetz
-------------
 04:05:06-08
(1 row)
```


Special Values

The special values supported by GaussDB are converted to common date/time values when being read. For details, see **Table 7-15**.

Table 7-15 Special values

Input String	Applicable Type	Description
epoch	date and timestamp	1970-01-01 00:00:00+00 (Unix system time zero)
infinity	timestamp	Later than any other timestamps
-infinity	timestamp	Earlier than any other timestamps
now	date, time, and timestamp	Start time of the current transaction
today	date and timestamp	Midnight today
tomorrow	date and timestamp	Midnight tomorrow
yesterday	date and timestamp	Midnight yesterday
allballs	time	00:00:00.00 UTC

Example:

```
-- Create a table.
gaussdb=# CREATE TABLE realtime_type_special(col1 varchar(20), col2 date, col3 timestamp, col4 time);

-- Insert data.
gaussdb=# INSERT INTO realtime_type_special VALUES('epoch', 'epoch', 'epoch', NULL);
gaussdb=# INSERT INTO realtime_type_special VALUES('now', 'now', 'now', 'now');
gaussdb=# INSERT INTO realtime_type_special VALUES('today', 'today', 'today', NULL);
gaussdb=# INSERT INTO realtime_type_special VALUES('tomorrow', 'tomorrow', 'tomorrow', NULL);
gaussdb=# INSERT INTO realtime_type_special VALUES('yesterday', 'yesterday', 'yesterday', NULL);

-- View data.
gaussdb=# SELECT * FROM realtime_type_special;
   col1    |        col2         |            col3            |      col4
-----------+---------------------+----------------------------+-----------------
 epoch     | 1970-01-01 00:00:00 | 1970-01-01 00:00:00        |
 now       | 2023-02-27 11:38:13 | 2023-02-27 11:38:13.032815 | 11:38:13.032815
 today     | 2023-02-27 00:00:00 | 2023-02-27 00:00:00        |
 tomorrow  | 2023-02-28 00:00:00 | 2023-02-28 00:00:00        |
 yesterday | 2023-02-26 00:00:00 | 2023-02-26 00:00:00        |
(5 rows)

gaussdb=# SELECT * FROM realtime_type_special WHERE col3 < 'infinity';
   col1    |        col2         |            col3            |      col4
-----------+---------------------+----------------------------+-----------------
 epoch     | 1970-01-01 00:00:00 | 1970-01-01 00:00:00        |
 now       | 2023-02-27 11:38:13 | 2023-02-27 11:38:13.032815 | 11:38:13.032815
 today     | 2023-02-27 00:00:00 | 2023-02-27 00:00:00        |
 tomorrow  | 2023-02-28 00:00:00 | 2023-02-28 00:00:00        |
 yesterday | 2023-02-26 00:00:00 | 2023-02-26 00:00:00        |
(5 rows)

gaussdb=# SELECT * FROM realtime_type_special WHERE col3 > '-infinity';
```



```
   col1    |        col2         |             col3             |      col4
-----------+---------------------+------------------------------+-----------------
 epoch     | 1970-01-01 00:00:00 | 1970-01-01 00:00:00          |
 now       | 2023-02-27 11:38:13 | 2023-02-27 11:38:13.032815 | 11:38:13.032815
 today     | 2023-02-27 00:00:00 | 2023-02-27 00:00:00          |
 tomorrow  | 2023-02-28 00:00:00 | 2023-02-28 00:00:00          |
 yesterday | 2023-02-26 00:00:00 | 2023-02-26 00:00:00          |
(5 rows)

gaussdb=# SELECT * FROM realtime_type_special WHERE col3 > 'now';
   col1   |        col2         |        col3         | col4
----------+---------------------+---------------------+------
 tomorrow | 2023-02-28 00:00:00 | 2023-02-28 00:00:00 |
(1 row)

gaussdb=# SELECT * FROM realtime_type_special WHERE col3 = 'today';
 col1  |        col2         |        col3         | col4
-------+---------------------+---------------------+------
 today | 2023-02-27 00:00:00 | 2023-02-27 00:00:00 |
(1 row)

gaussdb=# SELECT * FROM realtime_type_special WHERE col3 = 'tomorrow';
   col1   |        col2         |        col3         | col4
----------+---------------------+---------------------+------
 tomorrow | 2023-02-28 00:00:00 | 2023-02-28 00:00:00 |
(1 row)

gaussdb=# SELECT * FROM realtime_type_special WHERE col3 > 'yesterday';
   col1   |        col2         |             col3             |      col4
----------+---------------------+------------------------------+-----------------
 now      | 2023-02-27 11:38:13 | 2023-02-27 11:38:13.032815 | 11:38:13.032815
 today    | 2023-02-27 00:00:00 | 2023-02-27 00:00:00          |
 tomorrow | 2023-02-28 00:00:00 | 2023-02-28 00:00:00          |
(3 rows)

gaussdb=# SELECT TIME 'allballs';
   time
----------
 00:00:00
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE realtime_type_special;
```

Interval Input

The input of **reltime** can be any valid interval in text format. It can be a number (negative numbers and decimals are also allowed) or a specific time, which must be in SQL standard format or ISO-8601 format. In addition, the text input needs to be enclosed with single quotation marks ('').

For details about interval input, see **Table 7-16**.

Table 7-16 Interval input types

Input	Output	Description
60	2 mons	Numbers are used to indicate intervals. The default unit is day. Decimals and negative numbers are allowed. Particularly, a negative interval syntactically means how long before.
31.25	1 mons 1 days 06:00:00	
-365	-12 mons -5 days	
1 years 1 mons 8 days 12:00:00	1 years 1 mons 8 days 12:00:00	Intervals are in POSTGRES format. They can contain both positive and negative numbers and are case-insensitive. Output is a simplified POSTGRES interval converted from the input.
-13 months -10 hours	-1 years -25 days -04:00:00	
-2 YEARS +5 MONTHS 10 DAYS	-1 years -6 mons -25 days -06:00:00	
P-1.1Y10M	-3 mons -5 days -06:00:00	Intervals are in ISO-8601 format. They can contain both positive and negative numbers and are case-insensitive. Output is a simplified POSTGRES interval converted from the input.
-12H	-12:00:00	

Example:

```
-- Create a table.
gaussdb=# CREATE TABLE reltime_type_tab(col1 character(30), col2 reltime);

-- Insert data.
gaussdb=# INSERT INTO reltime_type_tab VALUES ('90', '90');
gaussdb=# INSERT INTO reltime_type_tab VALUES ('-366', '-366');
gaussdb=# INSERT INTO reltime_type_tab VALUES ('1975.25', '1975.25');
gaussdb=# INSERT INTO reltime_type_tab VALUES ('-2 YEARS +5 MONTHS 10 DAYS', '-2 YEARS +5
MONTHS 10 DAYS');
gaussdb=# INSERT INTO reltime_type_tab VALUES ('30 DAYS 12:00:00', '30 DAYS 12:00:00');
gaussdb=# INSERT INTO reltime_type_tab VALUES ('P-1.1Y10M', 'P-1.1Y10M');

-- View data.
gaussdb=# SELECT * FROM reltime_type_tab;
              col1              |                col2
--------------------------------+-------------------------------------
 90                             | 3 mons
 -366                           | -1 years -18:00:00
 1975.25                        | 5 years 4 mons 29 days
 -2 YEARS +5 MONTHS 10 DAYS     | -1 years -6 mons -25 days -06:00:00
 30 DAYS 12:00:00               | 1 mon 12:00:00
 P-1.1Y10M                      | -3 mons -5 days -06:00:00
(6 rows)

-- Drop the table.
gaussdb=# DROP TABLE reltime_type_tab;
```


7.3.7 Geometric Types

Table 7-17 lists the geometric types that can be used in GaussDB. The most fundamental type, the point, forms the basis for all of the other types.

Table 7-17 Geometric types

Name	Storage Space	Description	Representation
point	16 bytes	Point on a plane	(x,y)
lseg	32 bytes	Finite line segment	((x1,y1),(x2,y2))
box	32 bytes	Rectangle	((x1,y1),(x2,y2))
path	16 + 16*n* bytes	Closed path (similar to polygon)	((x1,y1),...)
path	16 + 16*n* bytes	Open path	[(x1,y1),...]
polygon	40 + 16*n* bytes	Polygon (similar to closed path)	((x1,y1),...)
circle	24 bytes	Circle	<(x,y),r> (center point and radius)

A rich set of functions and operators is available in GaussDB to perform various geometric operations, such as scaling, translation, rotation, and determining intersections. For details, see **Geometric Functions and Operators**.

Points

Points are the fundamental two-dimensional building block for geometric types. Values of the **point** type are specified using either of the following syntaxes:

```
( x , y )
x , y
```

x and **y** are the respective coordinates, as floating-point numbers. The value type of the points is float8.

Points are output using the first syntax.

Example:

```
gaussdb=# SELECT point(1.1, 2.2);
   point
-----------
 (1.1,2.2)
(1 row)
```

Line Segments

Line segments (**lseg**) are represented by pairs of points. Values of the **lseg** type are specified using any of the following syntaxes:


```
[ ( x1 , y1 ) , ( x2 , y2 ) ]
( ( x1 , y1 ) , ( x2 , y2 ) )
( x1 , y1 ) , ( x2 , y2 )
x1 , y1   ,   x2 , y2
```

(x1,y1) and **(x2,y2)** are the end points of the line segment. The value type of the points is float8.

Line segments are output using the first syntax.

Example:

```
gaussdb=# SELECT lseg(point(1.1, 2.2), point(3.3, 4.4));
        lseg
----------------------
 [(1.1,2.2),(3.3,4.4)]
(1 row)
```

Rectangles

Boxes are represented by pairs of points that are opposite corners of the box. Values of the **box** type are specified using any of the following syntaxes:

```
( ( x1 , y1 ) , ( x2 , y2 ) )
( x1 , y1 ) , ( x2 , y2 )
x1 , y1   ,   x2 , y2
```

(x1,y1) and **(x2,y2)** are any two opposite corners of the rectangle. The value type of the points is float8.

Rectangles are output using the second syntax.

Any two opposite corners can be supplied on input, but in this order, the values will be reordered as needed to store the upper right and lower left corners.

Example:

```
gaussdb=# SELECT box(point(1.1, 2.2), point(3.3, 4.4));
        box
---------------------
 (3.3,4.4),(1.1,2.2)
(1 row)
```

Paths

Paths are represented by lists of connected points. Paths can be open, where the first and last points in the list are considered not connected, or closed, where the first and last points are considered connected.

Values of the **path** type are specified using any of the following syntaxes:

```
[ ( x1 , y1 ) , ... , ( xn , yn ) ]
( ( x1 , y1 ) , ... , ( xn , yn ) )
( x1 , y1 ) , ... , ( xn , yn )
( x1 , y1   , ... ,   xn , yn )
x1 , y1   , ... ,   xn , yn
```

The points are the end points of the line segments comprising the path. The value type of the points is float8. Square brackets ([]) indicate an open path, while parentheses (()) indicate a closed path. When the outermost parentheses are omitted, as in the third through fifth syntax, a closed path is assumed.

Paths are output using the first or second syntax.

Example:

```
gaussdb=# SELECT path(polygon '((0,0),(1,1),(2,0))');
        path
--------------------
 ((0,0),(1,1),(2,0))
(1 row)
```

Polygons

Polygons are represented by lists of points (the vertexes of the polygon). Polygons are very similar to closed paths, but are stored differently and have their own set of support functions.

Values of the **polygon** type are specified using any of the following syntaxes:

```
( ( x1 , y1 ) , ... , ( xn , yn ) )
( x1 , y1 ) , ... , ( xn , yn )
( x1 , y1   , ... ,   xn , yn )
x1 , y1   , ... ,   xn , yn
```

A point indicates the vertex of a polygon. The value type of a point is float8.

Polygons are output using the first syntax.

Example:

```
gaussdb=# SELECT polygon(box '((0,0),(1,1))');
          polygon
---------------------------
 ((0,0),(0,1),(1,1),(1,0))
(1 row)
```

Circles

Circles are represented by a center point and radius. Values of the **circle** type are specified using the following syntax:

```
< ( x , y ) , r >
( ( x , y ) , r )
( x , y ) , r
x , y   , r
```

(x,y) is the center point and **r** is the radius of the circle. The value type of the points is float8.

Circles are output using the first syntax.

Example:

```
gaussdb=# SELECT circle(point(0,0),1);
  circle
-----------
 <(0,0),1>
(1 row)
```

7.3.8 Network Address Types

GaussDB offers data types to store IPv4, IPv6, and MAC addresses.

It is better to use these types instead of plain text types to store network addresses, because these types offer input error checking and specialized operators and functions (see **Network Address Functions and Operators**).

Table 7-18 Network address types

Name	Storage Space	Description
cidr	7 or 19 bytes	IPv4 or IPv6 networks
inet	7 or 19 bytes	IPv4 or IPv6 hosts and networks
macaddr	6 bytes	MAC address

When sorting inet or cidr data types, IPv4 addresses will always sort before IPv6 addresses, including IPv4 addresses encapsulated or mapped to IPv6 addresses, such as ::10.2.3.4 or ::ffff:10.4.3.2.

cidr

The cidr type (Classless Inter-Domain Routing) holds an IPv4 or IPv6 network address. The format for specifying networks is **address/y** where **address** is the network represented as an IPv4 or IPv6 address, and **y** is the number of bits in the netmask. If **y** is omitted, it is calculated using assumptions from the older classful network numbering system, except it will be at least large enough to include all of the octets written in the input.

Table 7-19 cidr type input examples

cidr Input	cidr Output	abbrev(cidr)
192.168.100.128/25	192.168.100.128/25	192.168.100.128/25
192.168/24	192.168.0.0/24	192.168.0/24
192.168/25	192.168.0.0/25	192.168.0.0/25
192.168.1	192.168.1.0/24	192.168.1/24
192.168	192.168.0.0/24	192.168.0/24
10.1.2	10.1.2.0/24	10.1.2/24
10.1	10.1.0.0/16	10.1/16
10	10.0.0.0/8	10/8
10.1.2.3/32	10.1.2.3/32	10.1.2.3/32
2001:4f8:3:ba::/64	2001:4f8:3:ba::/64	2001:4f8:3:ba::/64
2001:4f8:3:ba:2e0:81ff:fe22:d1f1/128	2001:4f8:3:ba:2e0:81ff:fe22:d1f1/128	2001:4f8:3:ba:2e0:81ff:fe22:d1f1
::ffff:127.0.0.0/120	::ffff:127.0.0.0/120	::ffff:127.0.0.0/120
::ffff:127.0.0.0/128	::ffff:127.0.0.0/128	::ffff:127.0.0.0/128

inet

The inet type holds an IPv4 or IPv6 host address, and optionally its subnet, all in one field. The subnet is represented by the number of network address bits present in the host address (the "netmask"). If the netmask is 32 and the address is an IPv4 address, then the value does not indicate a subnet, only a single host. In IPv6, because the address length is 128 bits, 128 bits specify a unique host address.

The input format for this type is **address/y** where **address** is an IPv4 or IPv6 address and **y** is the number of bits in the netmask. If **/y** is omitted, the subnet mask is **32** for an IPv4 address and **128** for an IPv6 address, and the value represents just a single host. On display, the **/y** portion is suppressed if the netmask specifies a single host.

The essential difference between the **inet** and **cidr** data types is that **inet** accepts values with nonzero bits to the right of the netmask, whereas **cidr** does not.

macaddr

The macaddr type stores MAC addresses, known for example from Ethernet card hardware addresses (although MAC addresses are used for other purposes as well). Input is accepted in the following formats:

```
'08:00:2b:01:02:03'
'08-00-2b-01-02-03'
'08002b:010203'
'08002b-010203'
'0800.2b01.0203'
'08002b010203'
```

These examples would all specify the same address. Upper and lower cases are accepted for the digits a through f. Output is always in the first of the forms shown.

7.3.9 Bit String Types

Bit strings are strings of 1's and 0's. They can be used to store bit masks.

GaussDB supports two bit string types: bit(n) and bit varying(n). Here, *n* is a positive integer. The maximum value of *n* is **83886080**, which is equivalent to 10 MB.

The bit type data must match the length *n* exactly. It is an error to attempt to store shorter or longer bit strings. Data of the bit varying type is of variable length up to the maximum length *n*. Longer strings will be rejected. Writing **bit** without a length is equivalent to **bit(1)**, while bit varying without a length specification means unlimited length.

NOTE

- If one explicitly casts a bit-string value to **bit(n)**, it will be truncated or zero-padded on the right to be exactly *n* bits, without raising an error.
- Similarly, if one explicitly casts a bit-string value to **bit varying(n)**, it will be truncated on the right if it is more than *n* bits.
- When the ADMS platform driver version 8.1.3-200 or earlier is used, use **::bit varying** to convert the bit type. Otherwise, an error may occur.


```
-- Create a table.
gaussdb=# CREATE TABLE bit_type_t1
(
    BT_COL1 INTEGER,
    BT_COL2 BIT(3),
    BT_COL3 BIT VARYING(5)
) ;

-- Insert data.
gaussdb=# INSERT INTO bit_type_t1 VALUES(1, B'101', B'00');

-- Specify the type length. An error is reported if an inserted string exceeds this length.
gaussdb=# INSERT INTO bit_type_t1 VALUES(2, B'10', B'101');
ERROR:  bit string length 2 does not match type bit(3)
CONTEXT:  referenced column: bt_col2

-- Specify the type length. Data is converted if it exceeds this length.
gaussdb=# INSERT INTO bit_type_t1 VALUES(2, B'10'::bit(3), B'101');

-- View data.
gaussdb=# SELECT * FROM bit_type_t1;
 bt_col1 | bt_col2 | bt_col3
---------+---------+---------
       1 | 101     | 00
       2 | 100     | 101
(2 rows)

-- Drop the table.
gaussdb=# DROP TABLE bit_type_t1;
```

7.3.10 UUID Type

The data type UUID stores Universally Unique Identifiers (UUID) as defined by RFC 4122, ISO/IEF 9834-8:2005, and related standards. This identifier is a 128-bit quantity that is generated by an algorithm chosen to make it very unlikely that the same identifier will be generated by anyone else in the known universe using the same algorithm.

A UUID is written as a sequence of lower-case hexadecimal digits, in several groups separated by hyphens, specifically a group of 8 digits followed by three groups of 4 digits followed by a group of 12 digits, for a total of 32 digits representing the 128 bits. An example of a UUID in this standard form is:

```
a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
```

GaussDB also accepts the following alternative forms for input: use of upper-case letters and digits, the standard format surrounded by braces, omitting some or all hyphens, adding a hyphen after any group of four digits. An example is provided as follows:

```
A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A11
{a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11}
a0eebc999c0b4ef8bb6d6bb9bd380a11
a0ee-bc99-9c0b-4ef8-bb6d-6bb9-bd38-0a11
```

Output is always in the standard form.

7.3.11 JSON/JSONB Types

JavaScript Object Notation (JSON) data can be a single scalar, an array, or a key-value pair object. The array and object can be called a container:

- Scalar: a number, Boolean, string, or null

- Array: defined in a pair of square brackets ([]), in which elements can be any type of JSON data, and are not necessarily of the same type.
- Object: defined in a pair of braces ({}), in which objects are stored in the format of key:value. Each key must be a string enclosed by a pair of double quotation marks (""), and its value can be any type of JSON data. In case of duplicate keys, the last key value will be used.

GaussDB offers two types for storing JSON data: JSON and JSONB. The JSON data type stores a complete copy of the input, retaining the entered spaces, duplicate keys, and sequence. The JSONB data type stores data in a decomposed binary form, removing semantic-irrelevant details and duplicate keys, and sorting key-values. Therefore, the JSONB data does not need to be parsed.

It can be found that both are of JSON type, and the same strings can be entered as input. The main difference between them is the efficiency. Because JSON data type stores an exact copy of the input text, the data must be parsed on every execution. In contrast, JSONB data is stored in its parsed binary form and can be processed faster, though this makes it slightly slower to input due to the conversion mechanism. In addition, because the JSONB data type is normalized, it supports more operations, for example, comparing sizes according to a specific rule. JSONB also supports indexing, which is a significant advantage.

Input Format

An input must be a JSON-compliant string, which is enclosed in single quotation marks ('').

null (null-json): The value can only be **null** in lowercase.

```
gaussdb=# SELECT 'null'::json;   -- suc
 json
------
 null
(1 row)

gaussdb=# SELECT 'NULL'::jsonb;  -- err
ERROR:  invalid input syntax for type json
```

Number (num-json): The value can be a positive or negative integer, decimal fraction, or 0. The scientific notation is supported.

```
gaussdb=# SELECT '1'::json;
json
------
 1
(1 row)

gaussdb=# SELECT '-1.5'::json;
 json
------
 -1.5
(1 row)

gaussdb=# SELECT '-1.5e-5'::jsonb, '-1.5e+2'::jsonb;
   jsonb   | jsonb
-----------+-------
 -0.000015 | -150
(1 row)

gaussdb=# SELECT '001'::json, '+15'::json, 'NaN'::json;  -- Redundant leading zeros, plus signs (+), NaN, and infinity are not supported.
ERROR:  invalid input syntax for type json
```


Boolean (bool-json): The value can only be **true** or **false** in lowercase.

```
gaussdb=# SELECT 'true'::json;
 json
------
 true
(1 row)

gaussdb=# SELECT 'false'::jsonb;
 jsonb
-------
 false
(1 row)
```

String (str-json): The value must be a string enclosed in double quotation marks ("").

```
gaussdb=# SELECT '"a"'::json;
 json
------
 "a"
(1 row)

gaussdb=# SELECT '"abc"'::jsonb;
 jsonb
-------
 "abc"
(1 row)
```

Array (array-json): Arrays are enclosed in square brackets ([]). Elements in the array can be any valid JSON data, and are unnecessarily of the same type.

```
gaussdb=# SELECT '[1, 2, "foo", null]'::json;
        json
--------------------
 [1, 2, "foo", null]
(1 row)

gaussdb=# SELECT '[]'::json;
 json
------
 []
(1 row)

gaussdb=# SELECT '[1, 2, "foo", null, [[]], {}]'::jsonb;
             jsonb
-------------------------------
 [1, 2, "foo", null, [[]], {}]
(1 row)
```

Object (object-json): The value is enclosed in braces ({}). The key must be a JSON-compliant string, and the value can be any valid JSON string.

```
gaussdb=# SELECT '{}'::json;
 json
------
 {}
(1 row)

gaussdb=# SELECT '{"a": 1, "b": {"a": 2,  "b": null}}'::json;
                json
------------------------------------
 {"a": 1, "b": {"a": 2,  "b": null}}
(1 row)

gaussdb=# SELECT '{"foo": [true, "bar"], "tags": {"a": 1, "b": null}}'::jsonb;
                       jsonb
-----------------------------------------------------
```



```
 {"foo": [true, "bar"], "tags": {"a": 1, "b": null}}
(1 row)
```

NOTICE

- Note that **'null'::json** and **null::json** are different, which are similar to the strings **str="null"** and **str=null**.
- For numbers, when scientific notation is used, JSONB expands them, while JSON stores an exact copy of the input text.

JSONB Advanced Features

- Precautions
  - It cannot be used as a partition key.
  - Foreign tables are not supported.

The main difference between JSON and JSONB lies in the storage mode. JSONB stores parsed binary data, which reflects the JSON hierarchy and facilitates direct access. Therefore, JSONB has many advanced features that JSON does not have.

- Format normalization
  - After the input object-json string is parsed into JSONB binary, semantically irrelevant details are naturally discarded, for example, spaces:

    ```
    gaussdb=# SELECT '    [1, " a ", {"a"    :1     }]  '::jsonb;
           jsonb
    ----------------------
     [1, " a ", {"a": 1}]
    (1 row)
    ```

  - For object-json, duplicate key-values are deleted and only the last key-value is retained. For example:

    ```
    gaussdb=# SELECT '{"a" : 1, "a" : 2}'::jsonb;
      jsonb
    ----------
     {"a": 2}
    (1 row)
    ```

  - For object-json, key-values will be re-sorted. The sorting rule is as follows: 1. Longer key-values are sorted last. 2. If the key-values are of the same length, the key-values with a larger ASCII code are sorted after the key-values with a smaller ASCII code:

    ```
    gaussdb=# SELECT '{"aa" : 1, "b" : 2, "a" : 3}'::jsonb;
              jsonb
    ----------------------------
    {"a": 3, "b": 2, "aa": 1}
    (1 row)
    ```

- Size comparison

  Format normalization ensures that only one form of JSONB data exists in the same semantics. Therefore, sizes may be compared according to a specific rule.

  - First, type comparison: **object-jsonb** > **array-jsonb** > **bool-jsonb** > **num-jsonb** > **str-jsonb** > **null-jsonb**
  - Content comparison if the data type is the same:

  - **str-jsonb**: The default text sorting rule of the database is used for comparison. A positive value indicates greater than, a negative value indicates less than, and **0** indicates equal.
  - **num-jsonb**: numeric comparison
  - **bool-jsonb**: **true** > **false**
  - **array-jsonb**: long elements > short elements. If the lengths are the same, compare each element in sequence.
  - **object-jsonb**: If the length of a key-value pair is longer than that of a short key-value pair, the keys are compared first, and then the values are compared.

  CAUTION

  For comparison within the **object-jsonb** type, the final result after format sorting is used for comparison. Therefore, the comparison result may not be intuitive compared with the direct input.

- Creating indexes, primary keys, and foreign keys
  - B-tree index

    B-tree indexes, primary keys, and foreign keys can be created for the JSONB type.
- Inclusion and existence

  Querying whether a JSON contains some elements or whether some elements exist in a JSON is an important capability of JSONB.

```
-- Simple scalar/primitive values contain only the identical value.
gaussdb=# SELECT '"foo"'::jsonb @> '"foo"'::jsonb;
 ?column?
----------
 t
(1 row)

-- The array on the left contains the character string on the right.
gaussdb=# SELECT '[1, "aa", 3]'::jsonb ? 'aa';
 ?column?
----------
 t
(1 row)

-- The array on the left contains all elements of the array on the right, regardless of the sequence and repetition.
gaussdb=# SELECT '[1, 2, 3]'::jsonb @> '[1, 3, 1]'::jsonb;
 ?column?
----------
 t
(1 row)

-- The object-json on the left contains all key-value pairs of object-json on the right.
gaussdb=# SELECT '{"product": "PostgreSQL", "version": 9.4, "jsonb":true}'::jsonb @>
'{"version":9.4}'::jsonb;
 ?column?
----------
 t
(1 row)
```



```
-- The array on the left does not contain all elements of the array on the right. The three elements on the left are 1, 2, and [1,3], but the elements on the right are 1 and 3.
gaussdb=# SELECT '[1, 2, [1, 3]]'::jsonb @> '[1, 3]'::jsonb;
 ?column?
----------
 f
(1 row)

gaussdb=# SELECT '{"foo": {"bar": "baz"}}'::jsonb @> '{"bar": "baz"}'::jsonb;
 ?column?
----------
 f
(1 row)
```

For details about the operators, see **JSON/JSONB Functions and Operators**.

- Functions and operators

  For details about the functions and operators supported by the JSON/JSONB type, see **JSON/JSONB Functions and Operators**.

7.3.12 HLL Type

HyperLoglog (HLL) is an approximation algorithm for efficiently counting the number of distinct values in a dataset. It features faster computing and lower space usage. You only need to store HLL data structures, instead of data sets. When new data is added to a dataset, make hash calculation on the data and insert the result to an HLL. Then, you can obtain the final result based on the HLL.

Table 7-20 compares HLL with other algorithms.

Table 7-20 Comparison between HLL and other algorithms

Item	Sorting Algorithm	Hash Algorithm	HLL
Time complexity	O(nlogn)	O(n)	O(n)
Space complexity	O(n)	O(n)	log(logn)
Error rate	0	0	≈0.8%
Storage space requirement	Size of original data	Size of original data	The maximum size is 16 KB by default.

HLL has advantages over others in the computing speed and storage space requirement. In terms of time complexity, the sorting algorithm needs O(nlogn) time for sorting, and the hash algorithm and HLL need O(n) time for full table scanning. In terms of storage space requirements, the sorting algorithm and hash algorithm need to store raw data before collecting statistics, whereas the HLL algorithm needs to store only the HLL data structures rather than the raw data, and thereby occupying a fixed space of about 16 KB.

NOTICE

- In the current default specifications, the maximum number of distinct values that can be calculated is about 1.1e + 15, and the error rate is 0.8%. If the calculation result exceeds the maximum, the error rate of the calculation result will increase, or the calculation will fail and an error will be reported.
- When using this feature for the first time, you need to evaluate the distinct values of the service, properly select configuration parameters, and perform verification to ensure that the accuracy meets requirements.
  - By default, the distinct value is 1.1e + 15. If the distinct value is NaN, you need to adjust log2m or use another algorithm to calculate the distinct value.
  - The hash algorithm has an extremely low probability of collision. However, you are still advised to select 2 or 3 hash seeds for verification when using the hash algorithm for the first time. If there is only a small difference between the distinct values, you can select any one of the seeds as the hash seed.

Table 7-21 describes main HLL data structures.

Table 7-21 Main HLL data structures

Data Type	Description
hll	The HLL header is a 27-byte column. By default, the data length ranges from 0 KB to 16 KB. The distinct value can be obtained.

When you create an HLL data type, 0 to 4 input parameters are supported. The parameter meanings and specifications are the same as those of the hll_empty function. The first parameter is **log2m**, indicating the logarithm of the number of buckets, and its value ranges from 10 to 16. The second parameter is **log2explicit**, indicating the threshold in explicit mode, and its value ranges from 0 to 12. The third parameter is **log2sparse**, indicating the threshold of the Sparse mode, and its value ranges from 0 to 14. The fourth parameter is **duplicatecheck**, indicating whether to enable duplicatecheck, and its value ranges from 0 to 1. When the input parameter is set to **–1**, the default value of the HLL parameter is used. You can run the **\d** or **\d+** command to view the parameters of the HLL type.

NOTE

When the HLL data type is created, the result varies depending on the input parameter behavior:

- When creating an HLL type, do not set the input parameter or set it to **–1**. Use the default value of the corresponding HLL parameter.
- If a valid value is set for the input parameter, the corresponding HLL parameter uses the input value.
- If the input value is invalid, an error is reported when the HLL type is created.

```
-- Create an HLL table without specifying input parameters.
gaussdb=# CREATE TABLE t1 (id integer, set hll);
```



```
gaussdb=# \d t1
      Table "public.t1"
 Column |  Type   | Modifiers
--------+---------+-----------
 id     | integer |
 set    | hll     |

-- Create an HLL table, specify the first two input parameters, and use the default values for the last two
input parameters.
gaussdb=# CREATE TABLE t2 (id integer, set hll(12,4));
gaussdb=# \d t2
          Table "public.t2"
 Column |      Type      | Modifiers
--------+----------------+-----------
 id     | integer        |
 set    | hll(12,4,12,0) |

-- Create an HLL table, specify the third input parameter, and use default values for other parameters.
gaussdb=# CREATE TABLE t3(id int, set hll(-1,-1,8,-1));
gaussdb=# \d t3
          Table "public.t3"
 Column |      Type      | Modifiers
--------+----------------+-----------
 id     | integer        |
 set    | hll(14,10,8,0) |

-- When a user creates an HLL table and specifies an invalid input parameter, an error is reported.
gaussdb=# CREATE TABLE t4(id int, set hll(5,-1));
ERROR:  log2m = 5 is out of range, it should be in range 10 to 16, or set -1 as default

-- Drop the created HLL table.
gaussdb=# DROP TABLE t1,t2,t3;
DROP TABLE
```

NOTE

When inserting an HLL object to an HLL table, ensure that the parameters of the HLL type are the same as those of the inserted object. Otherwise, an error is reported.

```
-- Create an HLL table.
gaussdb=# CREATE TABLE t1(id integer, set hll(14));

-- Insert an HLL object to a table. The insertion succeeds because parameter types are consistent.
gaussdb=# INSERT INTO t1 VALUES (1, hll_empty(14,-1));

-- Insert an HLL object to a table. The insertion fails because parameter types are inconsistent.
gaussdb=# INSERT INTO t1(id, set) VALUES (1, hll_empty(14,5));
ERROR:  log2explicit does not match: source is 5 and dest is 10

-- Drop the table.
gaussdb=# DROP TABLE t1;
```

The following describes HLL application scenarios:

- Scenario 1: "Hello World"

  The following example shows how to use the HLL data type:

  ```
  -- Create an HLL table.
  gaussdb=# CREATE TABLE helloworld (id integer, set hll);

  -- Insert an empty HLL to the table.
  gaussdb=# INSERT INTO helloworld(id, set) VALUES (1, hll_empty());

  -- Add a hashed integer to the HLL.
  gaussdb=# UPDATE helloworld SET set = hll_add(set, hll_hash_integer(12345)) WHERE id = 1;

  -- Add a hashed string to the HLL.
  gaussdb=# UPDATE helloworld SET set = hll_add(set, hll_hash_text('hello world')) WHERE id = 1;

  -- Obtain the number of distinct values of the HLL.
  ```



```
gaussdb=# SELECT hll_cardinality(set) FROM helloworld WHERE id = 1;
 hll_cardinality
-----------------
               2
(1 row)

-- Drop the table.
gaussdb=#  DROP TABLE helloworld;
```

- Scenario 2: Collect statistics about website visitors.

  The following example shows how an HLL collects statistics on the number of users visiting a website within a period of time:

```
-- Create a raw data table to show that a user has visited the website at a certain time:
gaussdb=# CREATE TABLE facts (
        date            date,
        user_id         integer
);

-- Create a raw data table to show that a user has visited the website at a certain time.
gaussdb=# INSERT INTO facts VALUES ('2019-02-20', generate_series(1,100));
gaussdb=# INSERT INTO facts VALUES ('2019-02-21', generate_series(1,200));
gaussdb=# INSERT INTO facts VALUES ('2019-02-22', generate_series(1,300));
gaussdb=# INSERT INTO facts VALUES ('2019-02-23', generate_series(1,400));
gaussdb=# INSERT INTO facts VALUES ('2019-02-24', generate_series(1,500));
gaussdb=# INSERT INTO facts VALUES ('2019-02-25', generate_series(1,600));
gaussdb=# INSERT INTO facts values ('2019-02-26', generate_series(1,700));
gaussdb=# INSERT INTO facts VALUES ('2019-02-27', generate_series(1,800));

-- Create another table and specify an HLL column:
gaussdb=# CREATE TABLE daily_uniques (
    date            date UNIQUE,
    users           hll
);

-- Group data by date and insert the data into the HLL:
gaussdb=# INSERT INTO daily_uniques(date, users)
    SELECT date, hll_add_agg(hll_hash_integer(user_id))
    FROM facts
    GROUP BY 1;

-- Calculate the numbers of users visiting the website every day.
gaussdb=# SELECT date, hll_cardinality(users) from daily_uniques order by date;
    date    | hll_cardinality
------------+------------------
 2019-02-20 |              100
 2019-02-21 | 200.217913059312
 2019-02-22 |  301.76494508014
 2019-02-23 | 400.862858326446
 2019-02-24 | 502.626933349694
 2019-02-25 | 601.922606454213
 2019-02-26 | 696.602316769498
 2019-02-27 | 798.111731634412
(8 rows)

-- Calculate the number of users who had visited the website in the week from February 20, 2019 to
February 26, 2019.
gaussdb=# SELECT hll_cardinality(hll_union_agg(users)) FROM daily_uniques WHERE date >=
'2019-02-20'::date AND date <= '2019-02-26'::date;
 hll_cardinality
------------------
 696.602316769498
(1 row)

-- Calculate the number of users who had visited the website yesterday but have not visited the
website today:
gaussdb=# SELECT date, (#hll_union_agg(users) OVER two_days) - #users AS lost_uniques FROM
daily_uniques WINDOW two_days AS (ORDER BY date ASC ROWS 1 PRECEDING);  -- The default
compatibility details (O compatibility) are listed as
follows:
```



```
        date         | lost_uniques
---------------------+--------------
 2019-02-20 00:00:00 |            0
 2019-02-21 00:00:00 |            0
 2019-02-22 00:00:00 |            0
 2019-02-23 00:00:00 |            0
 2019-02-24 00:00:00 |            0
 2019-02-25 00:00:00 |            0
 2019-02-26 00:00:00 |            0
 2019-02-27 00:00:00 |            0
(8 rows)

-- Drop the table.
gaussdb=# DROP TABLE facts;
gaussdb=# DROP TABLE daily_uniques;
```

- Scenario 3: The data to be inserted does not meet the requirements of the HLL data structure.

  When inserting data into a column of the HLL type, ensure that the data meets the requirements of the HLL data structure. If the data does not meet the requirements after being parsed, an error will be reported. In the following example, **'E\\1234'** to be inserted does not meet the requirements of the HLL data structure after being parsed. As a result, an error is reported.

```
gaussdb=# CREATE TABLE test(id integer, set hll);
gaussdb=# INSERT INTO test VALUES(1, 'E\\1234');
ERROR:  not a hll type, size=6 is not enough

gaussdb=# DROP TABLE test;
```

7.3.13 Range Types

A range type is a data type that represents the range of a value of an element type (called the *subtype* of a range). For example, the range of timestamp may be used to express a time range in which a conference room is reserved. The data type is tsrange (short for timestamp range), and timestamp is its subtype. The subtype must have an overall order so that the element value can be clearly specified within a range, before, or after.

Range types are useful because they can express multiple element values in a single range value and can clearly express concepts such as range overlapping. The time and date range used for scheduling is a good example, and the price range and the range of an instrument are also examples of range type.

Built-in Range

The following built-in ranges are available:

- int4range: integer range.
- int8range: bigint range.
- numrange: numeric range.
- tsrange: range of timestamp without the time zone.
- tstzrange: range of timestamp with the time zone
- daterange: date range.

In addition, you can customize range types. For details, see **CREATE TYPE**.

Example

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE reservation (room int, during tsrange);
gaussdb=# INSERT INTO reservation VALUES (1108, '[2010-01-01 14:30, 2010-01-01 15:30)');
-- Inclusion
gaussdb=# SELECT int4range(10, 20) @> 3;
 ?column?
----------
 f
(1 row)
-- Overlapping
gaussdb=# SELECT numrange(11.1, 22.2) && numrange(20.0, 30.0);
 ?column?
----------
 t
(1 row)
-- Upper bound extraction
gaussdb=# SELECT upper(int8range(15, 25));
 upper
-------
    25
(1 row)
-- Calculate the intersection.
gaussdb=# SELECT int4range(10, 20) * int4range(15, 25);
 ?column?
----------
 [15,20)
(1 row)
-- Is the range empty?
gaussdb=# SELECT isempty(numrange(1, 5));
 isempty
---------
 f
(1 row)
-- Drop the table.
gaussdb=# DROP TABLE reservation;
```

See the complete list of operators and functions on a range type in **Range Functions and Operators**.

Including and Excluding Bounds

Each non-empty range has two bounds, a lower bound and an upper bound. All values between the upper and lower bounds are included in the range. An inclusion bound means that the bound value itself is included in the range, while an exclusion bound means that the bound value is not included in the range.

In a textual form of a range, the inclusion lower bound is expressed as "[" and an exclusion lower bound is expressed as "(". Similarly, one including the upper bound is expressed as "]" and one excluding the upper bound is expressed as ")" (for details, see **Range Input/Output**).

The lower_inc and upper_inc functions test the upper and lower bounds of a range value, respectively.

Infinite (Unbounded) Range

When the lower bound of a range is unbounded, it means that all values less than the upper bound are included in the range. Similarly, when the upper bound of a range is unbounded, all values greater than the lower bound are included in the range. When both the upper and lower bounds are unbounded, all values of the element type are considered within the range. The missing bounds are

automatically converted to exclusions. You can think of these missing values as positive infinity or negative infinity, but they are special range type values and are considered to be positive and negative infinity values that go beyond any range element type.

Element types with the infinity values can be used as explicit bound values. For example, in the timestamp range, [today, infinity) does not include a special timestamp value infinity.

The lower_inf and upper_inf functions test the infinite upper and lower bounds of a range, respectively.

Range Input/Output

The input of a range value must follow one of the following formats:

```
(lower-bound, upper-bound)
(lower-bound, upper-bound]
[lower-bound, upper-bound)
[lower-bound, upper-bound]
Empty
```

The output of a range value must follow one of the following formats:

```
[lower-bound, upper-bound)
Empty
```

Parentheses () or square brackets [] indicate whether the upper and lower bounds are excluded or included. Note that the last format is empty, which represents an empty range (a range that does not contain values).

The value of *lower-bound* can be a valid input character string of the subtype or null, indicating that there is no lower bound. Similarly, *upper-bound* can be a valid input string of the subtype or null, indicating that there is no upper bound.

Each bound value can be referenced using the quotation marks("") character. This is necessary if a bound value contains parentheses (), square brackets [], commas (,), quotation marks (""), or backslashes (\), because otherwise those characters will be considered part of the range syntax. To put the quotation mark or backslash in a referenced bound value, put a backslash in front of it (and a pair of double quotation marks in its quoted bound value represents one quotation mark character, which is similar to the single quotation mark rule in SQL character strings). In addition, you can avoid referencing or use backslash escapes to protect all data characters, otherwise they will be used as part of the range syntax. If you want to write a bound value that is an empty string, write **""**, indicating infinite bounds.

Spaces are allowed before and after a range value, but any space between parentheses or square brackets is used as part of the upper or lower bound value (depending on the element type, the space may or may not represent a value).

Examples:

```
-- 3 is included, 7 is not included, and all values between 3 and 7 are included.
gaussdb=# SELECT '[3,7)'::int4range;
 int4range
-----------
 [3,7)
(1 row)
-- Neither 3 nor 7 is included, but all values between them are included.
gaussdb=# SELECT '(3,7)'::int4range;
```



```
 int4range
-----------
 [4,7)
(1 row)
-- Only value 4 is included.
gaussdb=# SELECT '[4,4]'::int4range;
 int4range
-----------
 [4,5)
(1 row)
-- Exclude any value (and will be normalized to empty).
gaussdb=# SELECT '[4,4)'::int4range;
 int4range
-----------
Empty
(1 row)
```

Constructing Range

Each range type has a constructor function with the same name. Using constructor functions is often more convenient than writing a range literal constant because it avoids extra references to bound values. Constructor functions accept two or three parameters. Two parameters form a range in the standard form, where the lower bound is included and the upper bound is excluded, and three parameters form a range according to the bound specified by the third parameter. The third parameter must be one of the following character strings: (), (], [), or []. For example:

```
-- The complete format is: lower bound, upper bound, and textual parameters indicating the inclusion/
exclusion of bounds.
gaussdb=# SELECT numrange(1.0, 14.0, '(]');
  numrange
------------
 (1.0,14.0]
(1 row)
-- If the third parameter is ignored, it is assumed to be '[)'.
gaussdb=# SELECT numrange(1.0, 14.0);
  numrange
------------
 [1.0,14.0)
(1 row)
-- Although '(]' is specified here, the value will be converted to the standard format when displayed,
because int8range is a discrete range type (see below).
gaussdb=# SELECT int8range(1, 14, '(]');
 int8range
-----------
 [2,15)
(1 row)
-- Using NULL for a bound causes the range to be unbounded on that side.
gaussdb=# SELECT numrange(NULL, 2.2);
 numrange
----------
 (,2.2)
(1 row)
```

Discrete Range

A range element type has a well-defined "step" such as integer or date. In these types, if there is no valid value between two elements, they can be called adjacent. This is in contrast to a continuous range in which other element values can always be identified between two given values. For example, a range above the numeric type is continuous, and the range of timestamp is also continuous. (Although timestamp has limited precision and can be considered as discrete in

theory, it can be considered as continuous because the step is not normally considered.)

Another way to consider discrete range types is to have a clear "next" or "previous" value for each element value. With this idea in mind, you can switch between inclusion and exclusion expressions of a range bound by replacing it with the original given next or previous element value. For example, in an integer range type, [4,8] and (3,9) represent the same set of values, but not for numeric ranges.

A discrete range type should have a regularization function that knows the expected step size of the element type. The regularization function can convert the equivalents of the range type to the same expression, in particular consistent with the inclusion or exclusion bounds. If you do not specify a regularization function, ranges with different formats will always be considered as unequal, even if they actually express the same set of values.

The built-in range types int4range, int8range, and daterange use a regularized form that includes the lower bound and excludes the upper bound, that is, [). However, user-defined range types can use other conventions.

Defining New Range

Users can define their own range types. A common reason is to use the range on the subtype that is not provided in the built-in range type. For example, to create the range type subtype float8, run the following command:

```
gaussdb=# CREATE TYPE floatrange AS RANGE (
    subtype = float8,
    subtype_diff = float8mi
);
gaussdb=# SELECT '[1.234, 5.678]'::floatrange;
  floatrange
---------------
 [1.234,5.678]
(1 row)
gaussdb=# DROP TYPE floatrange;
```

Because float8 does not have a meaningful "step", the regularization function is not defined in this example.

Defining your own range type also allows you to specify a different subtype B-tree operator class or collection to change the sort order to determine which values fall within the given range.

If the subtype is considered to have a discrete value instead of a continuous value, the **CREATE TYPE** command should specify a canonical function. The regularization function receives an input range value and must return an equivalent range value that may have different bounds and formats. For two ranges, for example, [1, 7] and [1, 8) that represent the same value set, the output must be the same. There is no relationship between choosing which expression to use as the regularization function, as long as two values of equal value in different formats can always be mapped to the same value in the same format. In addition to adjusting the inclusion/exclusion bound format, if the expected compensation is larger than the subtype can store, a regularization function may round the bound value. For example, a range type above a timestamp might be defined as having a one-hour epoch, so the regularization function might need to round off bounds that are not multiples of an hour, or might throw an error directly.

The subtype difference function uses two subtype input values and returns a difference expressed as a float8 value (X minus Y). In the example above, we can use functions under the regular float8 subtraction operator. However, for any other subtype, some type conversion may be required. There may also be a need for innovative ideas on how to express differences as numbers. For maximum extensibility, the subtype_diff function should agree with the sort order of the selected operator class and sort rules. That is, if the first parameter of the sort order is greater than the second parameter, the result should be a positive value.

The following is an example of the subtype_diff function:

```
gaussdb=# CREATE FUNCTION time_subtype_diff(x time, y time) RETURNS float8 AS 'SELECT
EXTRACT(EPOCH FROM (x - y))' LANGUAGE sql STRICT IMMUTABLE;
gaussdb=# CREATE TYPE timerange AS RANGE (
    subtype = time,
    subtype_diff = time_subtype_diff
 );
gaussdb=# SELECT '[11:10, 23:00]'::timerange;
     timerange
---------------------
 [11:10:00,23:00:00]
(1 row)
gaussdb=# DROP TYPE timerange;
gaussdb=# DROP FUNCTION time_subtype_diff;
```

For details about how to create a range type, see **CREATE TYPE**.

Index

In addition, B-tree indexes can be created on table columns of the range type. For these index types, basically the only useful range operation is equivalence. Using the corresponding < and > operators, there is a B-tree sort order for range value definitions, but that order is fairly arbitrary and is often less useful in the reality. The B-tree support for range types is primarily designed to allow sorting within a query, rather than creating an index.

7.3.14 Object Identifier Types

Object identifiers (OIDs) are used internally by GaussDB as primary keys for various system catalogs. OIDs are not added to user-created tables by the system. The OID type represents an object identifier.

The OID type is currently implemented as an unsigned four-byte integer. So, using a user-created table's OID column as a primary key is discouraged.

Table 7-22 Object identifier types

Name	Reference	Description	Example
OID	N/A	Numeric object identifier	564182
CID	N/A	Command identifier. This is the data type of the system columns **cmin** and **cmax**. Command identifiers are 32-bit quantities.	N/A

Name	Reference	Description	Example
XID	N/A	A transaction identifier. This is the data type of the system columns **xmin** and **xmax**. Transaction identifiers are 64-bit quantities.	N/A
TID	N/A	Row identifier. This is the data type of the system column **ctid**. A row ID is a pair (block number, tuple index within block) that identifies the physical location of the row within its table.	N/A
REGCONFIG	pg_ts_config	Text search configuration	english
REGDICTIONARY	pg_ts_dict	Text search dictionary	simple
REGOPER	pg_operator	Operator name	N/A
REGOPERATOR	pg_operator	Operator with argument types	*(integer,integer) or -(NONE,integer)
REGPROC	pg_proc	Function name	sum
REGPROCEDURE	pg_proc	Function with argument types	sum(int4)
REGCLASS	pg_class	Relation name	pg_type
REGTYPE	pg_type	Data type name	integer

The **OID** type is used for a column in the database system catalog.

Example:

```
gaussdb=# SELECT oid FROM pg_class WHERE relname = 'pg_type';
 oid
------
 1247
(1 row)
```

The alias type for **OID** is **REGCLASS** which allows simplified search for **OID** values.

Example:

```
gaussdb=# SELECT attrelid,attname,atttypid,attstattarget FROM pg_attribute WHERE attrelid =
'pg_type'::REGCLASS;
 attrelid |  attname   | atttypid | attstattarget
----------+------------+----------+---------------
```



```
 1247 | xc_node_id     |       23 |           0
 1247 | tableoid       |       26 |           0
 1247 | cmax           |       29 |           0
 1247 | xmax           |       28 |           0
 1247 | cmin           |       29 |           0
 1247 | xmin           |       28 |           0
 1247 | oid            |       26 |           0
 1247 | ctid           |       27 |           0
 1247 | typname        |       19 |          -1
 1247 | typnamespace   |       26 |          -1
 1247 | typowner       |       26 |          -1
 1247 | typlen         |       21 |          -1
 1247 | typbyval       |       16 |          -1
 1247 | typtype        |       18 |          -1
 1247 | typcategory    |       18 |          -1
 1247 | typispreferred |       16 |          -1
 1247 | typisdefined   |       16 |          -1
 1247 | typdelim       |       18 |          -1
 1247 | typrelid       |       26 |          -1
 1247 | typelem        |       26 |          -1
 1247 | typarray       |       26 |          -1
 1247 | typinput       |       24 |          -1
 1247 | typoutput      |       24 |          -1
 1247 | typreceive     |       24 |          -1
 1247 | typsend        |       24 |          -1
 1247 | typmodin       |       24 |          -1
 1247 | typmodout      |       24 |          -1
 1247 | typanalyze     |       24 |          -1
 1247 | typalign       |       18 |          -1
 1247 | typstorage     |       18 |          -1
 1247 | typnotnull     |       16 |          -1
 1247 | typbasetype    |       26 |          -1
 1247 | typtypmod      |       23 |          -1
 1247 | typndims       |       23 |          -1
 1247 | typcollation   |       26 |          -1
 1247 | typdefaultbin  |      194 |          -1
 1247 | typdefault     |       25 |          -1
 1247 | typacl         |     1034 |          -1
 1247 | typelemmod     |       23 |          -1
(39 rows)
```

7.3.15 Pseudo-Types

GaussDB type system contains a number of special-purpose entries that are collectively called pseudo-types. A pseudo-type cannot be used as a column data type, but it can be used to declare a function's argument or result type.

Each of the available pseudo-types is useful in situations where a function's behavior does not correspond to simply taking or returning a value of a specific SQL data type. **Table 7-23** lists all pseudo-types.

Table 7-23 Pseudo-types

Name	Description
any	Indicates that a function accepts any input data type.
anyelement	Indicates that a function accepts any data type.
anyarray	Indicates that a function accepts any array data type.
anynonarray	Indicates that a function accepts any non-array data type.
anyenum	Indicates that a function accepts any enum data type.

Name	Description
anyrange	Indicates that a function accepts any range data type.
cstring	Indicates that a function accepts or returns a null-terminated C string.
internal	Indicates that a function accepts or returns a server-internal data type.
language_handler	Indicates that a procedural language call handler is declared to return **language_handler**.
fdw_handler	Indicates that a foreign-data wrapper handler is declared to return **fdw_handler**.
record	Identifies a function returning an unspecified row type.
trigger	Indicates that a trigger function is declared to return **trigger**.
void	Indicates that a function returns no value.
opaque	Indicates an obsolete type name that formerly served all the above purposes.

Functions coded in C (whether built in or dynamically loaded) can be declared to accept or return any of these pseudo data types. It is up to the function author to ensure that the function will behave safely when a pseudo-type is used as an argument type.

Functions coded in procedural languages can use pseudo-types only as allowed by their implementation languages. At present the procedural languages all forbid use of a pseudo-type as argument type, and allow only **void** and **record** as a result type. Some also support polymorphic functions using the **anyelement**, **anyarray**, **anynonarray**, **anyenum**, and **anyrange** types.

Each location (parameter or return value) declared as the anyelement type is allowed to have any specific actual data type, but they must all be of the same actual type in any given query.

The internal pseudo-type is used to declare functions that are meant only to be called internally by the database system, and not by direct calling in an SQL query. If a function has at least one internal-type argument then it cannot be called from SQL. You are advised not to create any function that is declared to return internal unless it has at least one **internal** argument.

Example:

```
-- Create a table.
gaussdb=# CREATE TABLE t1 (a int);

-- Insert two data records.
gaussdb=# INSERT INTO t1 values(1),(2);

-- Create the showall() function.
gaussdb=# CREATE OR REPLACE FUNCTION showall() RETURNS SETOF record
AS $$ SELECT count(*) from t1; $$
```



```
LANGUAGE SQL;

-- Call the showall() function.
gaussdb=# SELECT showall();
 showall
---------
 (2)
(1 row)

-- Delete the function.
gaussdb=# DROP FUNCTION showall();

-- Delete the table.
gaussdb=# DROP TABLE t1;
```

7.3.16 XML Type

The XML data type can be used to store Extensible Markup Language (XML) data. The internal format of XML is the same as that of the TEXT data type. Its advantage over storing XML data in a TEXT field is that, XML data supports standard XML operation functions based on LIBXML2 and XML standardization check.

The XML data type can store well-formed documents as defined by the XML standard, as well as content fragments, which are defined by referencing broader "DOCUMENT NODE" XQuery and XPath data models. Roughly speaking, this means that there can be more than one top-level element or character node in a content fragment. The expression **XMLVALUE IS DOCUMENT** can be used to evaluate whether a particular XML value is a complete document or just a document fragment.

The XML parser converts an XML document into an XML DOM object. The document object model (DOM) defines standard methods for accessing and manipulating documents. XML DOM defines standard methods for accessing and manipulating XML documents. XML DOM views XML documents as a tree structure. All elements can be accessed through the DOM tree. You can modify or delete their contents and create new elements. Elements, their text, and their attributes are considered as nodes.

The XML bottom layer uses the same data structure as the TEXT type for storage. The maximum size is 1 GB.

Example:

```
gaussdb=# CREATE TABLE xmltest ( id int, data xml );
gaussdb=# INSERT INTO xmltest VALUES (1, 'one');
gaussdb=# INSERT INTO xmltest VALUES (2, 'two');
gaussdb=# SELECT * FROM xmltest ORDER BY 1;
 id | data
----+--------------------
1 | one
2 | two
(2 rows)
gaussdb=# SELECT xmlconcat(xmlcomment('hello'),
                           xmlelement(NAME qux, 'xml'),
                           xmlcomment('world'));
              xmlconcat
----------------------------------------
 <!--hello--><qux>xml</qux><!--world-->
(1 row)
gaussdb=# DROP TABLE xmltest;
```


NOTE

- The XML type does not support the following operations:
  - Logical expressions AND, OR, and NOT
  - Input parameter of a system function that is used as a non-XML operation function
  - Used as a distribution key, partition key, level-2 partition key, foreign key, primary key, or unique constraint.
  - Implicit conversion related to XML, including the conversion between strings and the XML data type
  - Array expression, row expression, subquery expression, TABLE OF, and TABLE OF INDEX
  - Use columns of the XML data format as common indexes, unique indexes, global indexes, local indexes, and partial indexes.
  - Comparison expressions >, <, >=, <=, =, <>, !=, ^=, <=>, BETWEEN AND, IS DISTINCT FROM, and IS NOT DISTINCT FROM
  - Condition expressions DECODE, NULLIF, GREATEST, and LEAST
  - Used as DISTINCT, GROUP BY, or ORDER BY parameters
  - Aggregate functions including sum, max, min, avg, list_agg, corr, covar_pop, covar_samp, stddev, stddev_pop, stddev_samp, var_pop, var_samp, variance, bit_and, bit_or, bool_and, bool_or, every, regr_avgx, regr_avgy, regr_count, regr_intercept, regr_r2, regr_slope, regr_sxx, regr_sxy, regr_syy, rank, and spread
  - ODBC-related APIs with binding parameters
- The XML type supports the following operations:
  - Physical backup and restoration
  - Comparison expressions IS NULL and IS NOT NULL
  - Condition expressions CASE and COALESCE
  - Global temporary tables and local temporary tables
  - Forcible type conversion
  - Expression indexes
  - Input XML values that comply with the XML standard
  - gs_dump and gs_restore
  - Parallel query, supporting the Astore and Ustore storage engines
  - Input parameters, output parameters, customized variables, and return values of a user-defined function
  - Input parameters, output parameters, customized variables, and return values of a stored procedure, as well as stored procedures that support autonomous transactions.
  - Character processing function quote_literal(string text) (explicitly converted to the character type) and quote_nullable(string text) (explicitly converted to the character type).
  - Aggregate functions count, array_agg, and checksum (explicitly converted to the character type), and string_agg (explicitly converted to the character type).
  - If addition, deletion, modification, and query of user-defined composite types involve XML types that are similar to XML columns in ordinary tables, the composite types must be inserted and modified based on the XML syntax.
  - JDBC and ODBC operations on XML data types are supported. The SELECT, UPDATE, INSERT, and DELETE operations can be performed on an XML column. You can enter an XML value using the SQL syntax and use the getSQLXML method of the ResultSet class to obtain the XML value. JDBC-related APIs with binding parameters are supported. For example, you can use the setSQLXML method in the PreparedStatement preprocessing API and the getSQLXML(int columnIndex) method in the ResultSet execution result set API.

In the calling process, use the java.sql.SQLXML API class to construct an XML object, set the specified object type to Oid.XML, and send the type ID and XML value to the server. After obtaining the result returned from the server, call ResultSet.getString. Then, use the java.sql.SQLXML API class to construct an XML object based on the obtained character string. In this case, the system checks whether the content complies with the XML standard again. Therefore, you can also use ResultSet.getString to directly obtain the XML string object.

7.3.17 XMLType

The XMLType data type is used to store XMLType data. Currently, data is stored in character strings in the internal format. Its advantage over storing XML data in a TEXT field is that, XMLTYPE data supports standard XML operation functions based on LIBXML2 and XML standardization check.

The XMLType type can store well-formed "documents" that comply with the XML standard.

The XML parser converts an XML document into an XML DOM object. The document object model (DOM) defines standard methods for accessing and manipulating documents. XML DOM defines standard methods for accessing and manipulating XML documents. XML DOM views XML documents as a tree structure. All elements can be accessed through the DOM tree. You can modify or delete their contents and create new elements. Elements, their text, and their attributes are considered as nodes. The maximum size is 1 GB.

Example:

```
gaussdb=# CREATE TABLE xmltypetest(id int, data xmltype);
gaussdb=# INSERT INTO xmltypetest VALUES (1, '<ss/>');
gaussdb=# INSERT INTO xmltypetest VALUES (2, '<xx/>');
gaussdb=# SELECT * FROM xmltypetest ORDER BY 1;
 id | data
----+-------
  1 | <ss/>
  2 | <xx/>
(2 rows)
gaussdb=# DROP TABLE xmltypetest;
```


NOTE

- The XMLType type does not support the following operations:
  - Logical expressions AND, OR, and NOT
  - Input parameter of a system function that is used as a non-XMLType operation function
  - Used as a distribution key, partition key, level-2 partition key, foreign key, primary key, or unique constraint.
  - Implicit conversion related to XMLType, including the conversion between strings and the XMLType data type
  - Array expression, row expression, subquery expression, TABLE OF, and TABLE OF INDEX
  - Use columns of the XMLType data format as common indexes, unique indexes, global indexes, local indexes, and partial indexes.
  - Comparison expressions >, <, >=, <=, =, <>, !=, ^=, <=>, BETWEEN AND, IS DISTINCT FROM, and IS NOT DISTINCT FROM
  - Condition expressions DECODE, NULLIF, GREATEST, and LEAST
  - Used as DISTINCT, GROUP BY, or ORDER BY parameters
  - Aggregate functions sum, max, min, avg, list_agg, corr, covar_pop, covar_samp, stddev, stddev_pop, stddev_samp, var_pop, var_samp, variance, bit_and, bit_or, bool_and, bool_or, every, regr_avgx, regr_avgy, regr_count, regr_intercept, regr_r2, regr_slope, regr_sxx, regr_sxy, regr_syy, rank, and spread
  - ODBC-related APIs with binding parameters
- The XMLType type supports the following operations:
  - Physical backup and restoration
  - Comparison expressions IS NULL and IS NOT NULL
  - Condition expressions CASE and COALESCE
  - Global temporary tables and local temporary tables
  - Forcible type conversion
  - Expression indexes
  - Input XMLType values that comply with the XML standard
  - gs_dump and gs_restore
  - Parallel query. The Astore and Ustore storage engines are supported.
  - Input parameters, output parameters, customized variables, and return values of a user-defined function
  - Input parameters, output parameters, customized variables, and return values of a stored procedure, as well as stored procedures that support autonomous transactions.
  - Character processing function quote_literal(string text) (explicitly converted to the character type) and quote_nullable(string text) (explicitly converted to the character type)
  - Aggregate functions count, array_agg, and checksum (explicitly converted to the character type), and string_agg (explicitly converted to the character type)
  - If addition, deletion, modification, and query of user-defined composite types involve XMLType types that are similar to XMLType columns in ordinary tables, the composite types must be inserted and modified based on the XMLType syntax.
  - The SELECT, UPDATE, INSERT, and DELETE operations can be performed on an XMLType column. You can enter an XMLType value using the SQL syntax.
- You can create a schema named **xmltype**. In the schema, you can create functions, but cannot use schema.func() to call functions defined in the schema.

7.3.18 Data Types Used by the Ledger Database

The ledger database uses the hash16 data type to store row-level hash digests or table-level hash digests, and uses the hash32 data type to store global hash digests or history table verification hashes.

Table 7-24 Hash type of the ledger database

Name	Description	Storage Space	Range
HASH16	Stored as an unsigned 64-bit integer.	8 bytes	0 to +18446744073709551615
HASH32	Stored as an unsigned integer array of 16 elements.	16 bytes	Value range of an unsigned integer array of 16 elements

The hash16 data type is used to store row-level or table-level hash digests in the ledger database. After obtaining the hash sequence of a 16-character hexadecimal string, the system invokes the hash16in function to convert the sequence into an unsigned 64-bit integer and stores the integer in a hash16 variable. For example:

```
Hexadecimal string: e697da2eaa3a775b; 64-bit unsigned integer: 16615989244166043483
Hexadecimal string: ffffffffffffffff; 64-bit unsigned integer: 18446744073709551615
```

The hash32 data type is used to store the global hash digest or history table verification hash in the ledger database. After obtaining the hash sequence of a 32-character hexadecimal string, the system invokes the hash32in function to convert the sequence to an array containing 16 unsigned integer elements. For example:

```
Hexadecimal string: 685847ed1fe38e18f6b0e2b18c00edee
Hash32 array: [104,88,71,237,31,227,142,24,246,176,226,177,140,0,237,238]
```

7.3.19 SET Type

The SET type is a collection type that contains string members and is defined when a table column is created.

Specifications

1. The number of members of the SET type ranges from 1 to 64. It cannot be defined as an empty set.
2. A member name can contain a maximum of 255 characters. An empty string can be used as a member name. The member name must be a character constant but cannot be a character constant obtained after calculation, for example, SET('a' || 'b', 'c').
3. The member name cannot contain commas (,) and must be unique.
4. Arrays and domain types of the SET type cannot be created.
5. The SET type is supported only when **sql_compatibility** is set to **B**.

6. The SET type cannot be used as the partition key of a partitioned table.
7. You need to use CASCADE to drop the SET type, and the associated table columns are also dropped.
8. For a Ustore table, if the table contains columns of the SET type and the recycle bin function is enabled, the table is directly deleted instead of being moved to the recycle bin.
9. ALTER TABLE cannot be used to change a column of the SET type to another SET type.
10. When a table or a table column associated with the SET type is deleted, or a table column of the SET type is changed to another type, the SET data type is also deleted.
11. CREATE TABLE { AS | LIKE } cannot be used to create a table containing the SET type.
12. The SET type is created with table columns, and its name is a combination of column names. If a data type with the same name already exists in the schema, the SET type fails to be created.
13. The SET type can be compared with the int2, int4, int8, and text types using =, <, >, <, <=, >, and >=.
14. The SET type can be converted to the int2, int4, int8, float4, float8, numeric, char, varchar, text and nvarchar2 data types.

Precautions

- The table column values of the SET type must be a subset of the set defined by the SET type. Example:
```
gaussdb=# CREATE TABLE employee (
  name text,
  site SET('beijing','shanghai','nanjing','wuhan')
);
```
- The value of the **site** column must be a subset of the preceding defined set and can be an empty set. If the provided value does not exist in the members of the defined set, an error is reported. Example:
```
gaussdb=# INSERT INTO employee values('zhangsan', 'nanjing,beijing');
INSERT 0 1
gaussdb=# INSERT INTO employee VALUES ('zhangsan', 'hangzhou');
ERROR:  invalid input value for set employee_site_set: 'hangzhou'
```
- Regardless of the sequence of the member values provided by the user, the queried values of the SET type are displayed in the defined sequence after the INSERT operation is successful.
```
gaussdb=# SELECT * FROM employee;
   name   |      site
----------+-----------------
 zhangsan | beijing,nanjing
(1 rows)
```
- The SET type is stored in bitmap mode. Members of the SET type are assigned different values according to the sequence in which they are defined. For example, the values of SET('beijing','shanghai','nanjing','wuhan') are as follows:

Table 7-25 Set members and their values

Set Member	Member Value	Binary Value
'beijing'	1	0001
'shanghai'	2	0010
'nanjing'	4	0100
'wuhan'	8	1000

Therefore, if a numeric value is assigned to a column of the SET type, the value is converted to the corresponding subset. For example, the binary value corresponding to **9** is **1001**, and the corresponding subset is **'beijing,wuhan'**.

```
gaussdb=# INSERT INTO employee values('lisi', 9);
INSERT 0 1
gaussdb=# SELECT * FROM employee;
   name   |      site
----------+-----------------
 zhangsan | beijing,nanjing
 lisi     | beijing,wuhan
(2 rows)
gaussdb=# DROP TABLE employee;
```

7.3.20 ACLItem Type

The aclitem data type is used to store object permission information. Its internal implementation is of the int type and supports the *user1=privs|user2* format.

The aclitem[] data type is an array consisting of ACL items. The supported format is {*user1 = privs1|user3*, *user2 = privs2|user3*}.

In the preceding information, *user1*, *user2*, and *user3* indicate the existing users or roles in the database, and *privs* indicates the permissions supported by the database. For details, see **Table 12-74**.

Example:

```
-- Create a user.
gaussdb=# CREATE USER user1 WITH PASSWORD 'Aa123456789';
gaussdb=# CREATE USER user2 WITH PASSWORD 'Aa123456789';
gaussdb=# CREATE USER omm WITH PASSWORD 'Aa123456789';

-- Create a data table table_acl that contains three columns of the int, aclitem, and aclitem[] types.
gaussdb=# CREATE TABLE table_acl (id int,priv aclitem,privs aclitem[]);
-- Insert a data record whose content is (1,'user1=arw/omm','{omm=d/user2,omm=w/omm}') into the table_acl table.
gaussdb=# INSERT INTO table_acl VALUES (1,'user1=arw/omm','{omm=d/user2,omm=w/omm}');
-- Insert a data record whose content is (2,'user1=aw/omm','{omm=d/user2}') into the table_acl table.
gaussdb=# INSERT INTO table_acl VALUES (2,'user1=aw/omm','{omm=d/user2}');
gaussdb=# SELECT * FROM  table_acl;
 id |     priv      |          privs
----+---------------+-------------------------
  1 | user1=arw/omm | {omm=d/user2,omm=w/omm}
  2 | user1=aw/omm  | {omm=d/user2}
(2 rows)

-- Drop the table and user.
gaussdb=# DROP USER user1;
gaussdb=# DROP USER user2;
```



```
gaussdb=# DROP USER omm;
gaussdb=# DROP TABLE table_acl;
```

7.3.21 Array Types

Array types can be used to store several elements of the same type.

Array Type Definition

Generally, an array data type is named by adding square brackets ([]) to the end of the data type name of an array element.

Example 1: Create a table named **sal_emp**. The table has the following columns: **name** of the text type, indicating employee names; **pay_by_quarter** of the integer element type, indicating an array of quarterly salaries; **phone_numbers** of the varchar(11) element type, indicating an array of mobile numbers.

```
gaussdb=# CREATE TABLE sal_emp (
name            text,
 pay_by_quarter  integer[],
phone_numbers   varchar(11)[]
gaussdb=# );
gaussdb=# DROP TABLE sal_emp;
```

Example 2: Define an array type in other ways. For details about the definition method and behavior, see the comments in the example.

```
gaussdb=# CREATE TABLE sal_emp (
name             text,
 pay_by_quarter1  integer[][],      -- Two-dimensional array of the int type.
 pay_by_quarter2  integer[3],       -- One-dimensional array of the int type. The size is 3.
 pay_by_quarter3  integer[3][3],    -- Two-dimensional array of the int type. The size of each dimension is 3.
 pay_by_quarter4  integer ARRAY,    -- One-dimensional array of the int type.
 pay_by_quarter5  integer ARRAY[3]  -- One-dimensional array of the int type. The size is 3.
);
gaussdb=# DROP TABLE sal_emp;
```

CAUTION

- The definition of the number of dimensions of an array does not take effect (which does not affect the runtime behavior). You are advised to use the method described in example 1 to define the array type and do not use multidimensional array data.
- The definition of the array size does not take effect (which does not affect the runtime behavior). You are advised to use the method described in example 1 to define the array type.
- The maximum number of dimensions of an array is 6.
- The restrictions on the number of array elements are as follows:
  1. The maximum number of elements is 134217727.
  2. The maximum storage space of all elements cannot exceed 1 GB minus 1 byte, that is, 1073741823 bytes.

Array Constructor

An array constructor is an expression that can build array values. A simple array constructor consists of the keyword ARRAY and an expression list of array element

values which are separated by commas and enclosed (,) by square brackets ([]). For example:

```
gaussdb=# SELECT ARRAY[1, 2, 3 + 4];
  array
---------
 {1,2,7}
(1 row)
```

By default, the element type of an array is the common type of member expressions and is determined by the same rules as the UNION or CASE structure (**UNION, CASE, and Related Constructs**). You can construct an array to the desired data type through explicit type conversion. The following is an example:

```
gaussdb=# SELECT ARRAY[1, 2, 3]::varchar[];
  array
---------
 {1,2,3}
(1 row)

gaussdb=# SELECT ARRAY['a', 'b', 'c'];
  array
---------
 {a,b,c}
(1 row)

gaussdb=# SELECT ARRAY['a', 'b', 'c']::int[];
ERROR:  invalid input syntax for integer: "a"
LINE 1: select ARRAY['a', 'b', 'c']::int[];
                ^
CONTEXT:  referenced column: array

gaussdb=# SELECT ARRAY[1::int, 'b', 'c'];
ERROR:  invalid input syntax for integer: "b"
LINE 1: select ARRAY[1::int, 'b', 'c'];
                             ^
CONTEXT:  referenced column: array
```

In addition to the preset basic types, the record type and table type can also be defined as array types. The following is an example:

```
gaussdb=# CREATE TYPE rec IS (c1 int, c2 int);
gaussdb=# SELECT ARRAY[(1, 1), (2, 2)]::rec[];
       array
-------------------
 {"(1,1)","(2,2)"}
(1 row)

gaussdb=# SELECT ARRAY[rec(1, 1), rec(2, 2)];
       array
-------------------
 {"(1,1)","(2,2)"}
(1 row)

gaussdb=# CREATE TABLE tab (c1 int, c2 int);
gaussdb=# SELECT ARRAY[(1, 1), (2, 2)]::tab[];
       array
-------------------
 {"(1,1)","(2,2)"}
(1 row)

gaussdb=# DROP TYPE rec;
gaussdb=# DROP TABLE tab;
```

An array must have a type. Therefore, when constructing an empty array, you must construct it as the required type. For example:

```
gaussdb=# SELECT ARRAY[]::int[];
 array
```



```
-------
 {}
(1 row)
```

You can also construct an array from the result of a subquery. In this case, the array constructor consists of the keyword ARRAY and a subquery enclosed in parentheses. The subquery must return only one separate column. The generated one-dimensional array generates an element for each row of the result in the subquery. The element type matches the output column of the subquery. For example:

```
gaussdb=# SELECT ARRAY(select generate_series(1, 6));
    array
---------------
 {1,2,3,4,5,6}
(1 row)
```

Multidimensional array values can be made by nesting array constructors. The ARRAY keyword in the inner constructor can be omitted. For example, the following two examples show the same result:

```
gaussdb=# SELECT ARRAY[ARRAY[1,2], ARRAY[3,4]];
    array
---------------
 {{1,2},{3,4}}
(1 row)

gaussdb=# SELECT ARRAY[[1,2], [3,4]];
    array
---------------
 {{1,2},{3,4}}
(1 row)
```

NOTE

- Inner constructors at the same layer must generate subarrays of the same dimension.
- Any type conversion applied to the outer ARRAY constructor is automatically applied to all inner constructors.

String Inputs of the Array Type

To write an array value as a literal constant (constant input), enclose the element values with braces and separate them with commas. The general format of an array constant is as follows:

```
'{ val1 delim val2 delim ... }'
```

In the preceding format, **delim** indicates the delimiter of the element type and is recorded in the **typdelim** column in the pg_type catalog of the type. Each **val** can be a constant of the array element type or a subarray. For example:

```
gaussdb=# SELECT '{1, 2, 3}'::int[] AS RESULT;
 result
---------
 {1,2,3}
(1 row)

gaussdb=# SELECT '{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}'::int[] AS RESULT;
         result
---------------------------
 {{1,2,3},{4,5,6},{7,8,9}}
(1 row)
```


Double quotation marks can be used around any element value, and double quotation marks must be used when the element value contains special characters such as commas or braces. The following is an example:

```
gaussdb=# SELECT '{" ", "NULL", null, "\\", "{", "}", ","}'::varchar[] AS RESULT;
               result
-----------------------------------
 {" ","NULL",NULL,"\\","{","}",","}
(1 row)

-- This example indicates that there is an array of the varchar type and there are seven varchar elements in total. The elements are as follows:
 1. A character string containing a space. 2. A character string containing "NULL". 3. A character string containing NULL. 4. A character string containing a backslash (\).
 5. A character string containing a left brace ({). 6. A character string containing a right brace (}). 7. A character string containing a comma (,).
```

NOTE

- For array string constant inputs, if the array element value is an empty string, contains braces, delimiters, double quotation marks, backslashes, or spaces, or matches the keyword NULL, then element inputs are enclosed in double quotation marks. An additional backslash is input if the element value contains double quotation marks or backslashes.
- The keyword NULL is case-insensitive.
- The input spaces not enclosed in double quotation marks are automatically skipped.
- You are advised to use the array constructor instead of character constants to construct array data.

String Outputs of the Array Type

The output of an array value consists of the output of the array element type plus some modifiers indicating the array structure. These modifiers consist of braces ({}) around array values plus delimiters between adjacent items. In a multidimensional array, each dimension has its own level of braces and contains delimiters between adjacent braces at the same level.

The data of the array type contains special characters (in the following description). The following is an example of character string output:

```
gaussdb=# SELECT ARRAY['{', '}', 'hello, world', '"', '\', '  ', NULL] AS RESULT;
                  array
-----------------------------------------------
 {"{","}","hello, world","\"","\\","  ",NULL}
(1 row)
```

NOTE

For array string constant outputs, if the array element value is an empty string or contains braces, delimiters, double quotation marks, backslashes, or spaces, or the element is NULL, then element outputs are enclosed in double quotation marks. An additional backslash is output if the element value contains double quotation marks or backslashes. This parameter corresponds to the string constant inputs.

Use of Array Types

The following is an example of using the array type:

```
-- Create a table with array columns and insert some data.
gaussdb=# CREATE TABLE orders (
 name  varchar,
```



```
items varchar[]
);
gaussdb=# INSERT INTO orders VALUES('a', ARRAY['Apple', 'Orange', 'Pear']);
gaussdb=# INSERT INTO orders VALUES('b', ARRAY['Mineral water', 'Coke', 'Sprite']);
gaussdb=# INSERT INTO orders VALUES('c', ARRAY['Mouse', 'Keyboard', 'Earphone']);
gaussdb=# INSERT INTO orders VALUES('d', '{Cabbage, Potato, Eggplant}');

-- Query data.
gaussdb=# SELECT * FROM orders ORDER BY name;
 name |       items
------+--------------------
 a    | {Apple,Orange,Pear}
 b    | {Mineral water,Coke,Sprite}
 c    | {Mouse,Keyboard,Earphone}
 d    | {Cabbage,Potato,Eggplant}
(4 rows)

-- Access array elements.
gaussdb=# SELECT items[1] FROM orders ORDER BY name;
 items
--------
 Apple
 Mineral water
 Mouse
 Cabbage
(4 rows)

-- If the accessed element exceeds the range or the accessed index is NULL, NULL is returned.
gaussdb=# SELECT items[4] FROM orders ORDER BY name;
 items
-------




(4 rows)

gaussdb=# SELECT items[null] FROM orders ORDER BY name;
 items
-------




(4 rows)

-- Access a subarray.
gaussdb=# SELECT items[1:2] FROM orders ORDER BY name;
     items
---------------
 {Apple,Orange}
 {Mineral water,Coke}
 {Mouse,Keyboard}
 {Cabbage,Potato}
(4 rows)

-- Updates the entire array.
gaussdb=# UPDATE orders SET items = ARRAY['Banana', 'Watermelon', 'Strawberry'] WHERE name = 'a';
gaussdb=# SELECT items FROM orders WHERE name = 'a';
      items
------------------
 {Banana,Watermelon,Strawberry}
(1 row)

-- Updates the elements of an array.
gaussdb=# UPDATE orders SET items[1] = 'Mango' WHERE name = 'a';
gaussdb=# SELECT items FROM orders WHERE name = 'a';
      items
------------------
```



```
 {Mango,Watermelon,Strawberry}
(1 row)

-- Updates the element fragment of an array.
gaussdb=# UPDATE orders SET items[1:2] = ARRAY['Computer','Mobile phone'] WHERE name = 'c';
gaussdb=# SELECT items FROM orders WHERE name = 'c';
      items
------------------
 {Computer,Mobile phone,Earphone}
(1 row)

-- Add an array element. All unassigned elements between the last element of the original array and the new element are set to NULL.
gaussdb=# UPDATE orders SET items[4] = 'Display' WHERE name = 'c';
gaussdb=# SELECT items FROM orders WHERE name = 'c';
          items
------------------------
 {Computer,Mobile phone,Earphone,Display}
(1 row)

gaussdb=# UPDATE orders SET items[6] = 'Display 2' WHERE name = 'c';
gaussdb=# SELECT items FROM orders WHERE name = 'c';
                items
--------------------------------------
 {Computer,Mobile phone,Earphone,Display,NULL,Display 2}
(1 row)
```

7.3.22 Vector Data Types

Vector data types include floatvector and boolvector.

- The floatvector data type indicates that multidimensional data contains data of the float type, for example, [1.0,3.0,11.0,110.0,62.0,22.0,4.0].

  NOTE

  - The floatvector member supports only single precision. Single-precision range: -3.402E+38 to +3.402E+38, 6-digit decimal digits.
  - The floatvector does not support NULL, Nan, or Inf as elements. If a vector contains NULL values, the database reports an error.
  - The floatvector cannot be NULL. The database reports an error when a NULL value is inserted, updated, or converted as vector data.

- The boolvector data type indicates that the multidimensional data contains data of the Boolean type, for example, [0,0,0,0,0,1,0,0,1,0,0,0].

  NOTE

  - Members of the boolvector data type can express Boolean data in **t(T)**/**f(F)**, **y(Y)**/**n(N)**, **1**/**0**, **true**/**false**, **yes**/**no**, or **on**/**off** mode.
  - The boolvector does not support NULL, Nan, or Inf as elements. If a vector contains NULL values, the database reports an error.
  - The boolvector cannot be NULL. The database reports an error when a NULL value is inserted, updated, or converted as vector data.

Usage of Vector Types

An example of using the vector type is as follows:

```
-- Create a table that contains vector types and set data dimensions. Dimensions must be specified for vector types during table creation.
gaussdb=# CREATE TABLE t1(id int unique, repr floatvector(4));
gaussdb=# CREATE TABLE t2(id int unique, repr boolvector(3));
-- Insert data.
```



```
gaussdb=# INSERT INTO t1 VALUES(0, '[30,12,12,25]');
gaussdb=# INSERT INTO t2 VALUES(1, '[1, 0, 1]');
```

7.4 Character Sets and Collations

A character set provides character encoding rules, and a collation provides character sorting rules. This section describes the character sets and collations in B-compatible GaussDB (**sql_compatibility = 'B'**). The following character sets, collation rules, and syntax are supported only in B-compatible mode:

For details about the character sets supported by GaussDB, see "ENCODING" in **CREATE DATABASE**. For details about the supported collations, see the PG_COLLATION system catalog.

Some character sets have default collations in B-compatible mode. For details, see **Table 7-220**.

The character set and collations are described as follows:

- Each character set has one or more collations and has only one default collation.
- Each collation has only one associated character set.
- The sorting results of the same data using different collations may be different.
- In GaussDB, utf8mb4 and utf8 are the same character set.
- When **sql_compatibility** is set to **'B'**, the BINARY and SQL_ASCII character sets are the same.
- You are advised to select the same character set for table columns and server_encoding to avoid performance loss caused by transcoding.

GaussDB supports the following functions:

- Multiple character sets can be used to store character strings.
- Collations can be used to compare character strings.
- Database-level, schema-level, table-level, and column-level character sets and collations are supported.

  NOTE

  Character strings with different character sets and collations cannot be used in the same server, database, table, or SQL statement.

7.4.1 Character Sets and Collations of the Client Connection

When data containing character set attributes is transmitted between the server and client, encoding conversion is automatically performed. After receiving an SQL statement from a client, the server converts the character set of the SQL statement from the client character set client_encoding to the database character set server_encoding. Before the query result data is sent to the client, the data is encoded and converted to the character_set_results character set of the client.

System parameters:

- server_encoding

Character set specified during database creation. For details, see **CREATE DATABASE**.

- client_encoding

  Character set of the client, which can be modified using the SET NAMES statement. For details, see **SET**. For details about the parameter, see **client_encoding** in "Configuring Running Parameters > GUC Parameters > Default Settings of Client Connection > Locale and Formatting" in *Administrator Guide*.

- character_set_connection

  Default character set of string constants for which no character sets are specified in SQL statements. For details about the parameter, see **character_set_connection** in "Configuring Running Parameters > GUC Parameters > Version and Platform Compatibility > Platform and Client Compatibility" in *Administrator Guide*.

- collation_connection

  Default collation of string constants for which no collations are specified in SQL statements. For details about the parameter, see **collation_connection** in "Configuring Running Parameters > GUC Parameters > Version and Platform Compatibility > Platform and Client Compatibility" in *Administrator Guide*.

- character_set_results

  Character set of the returned result. For details about the parameters, see **character_set_results** in "Configuring Running Parameters > GUC Parameters > Version and Platform Compatibility > Platform and Client Compatibility" in *Administrator Guide*.

  NOTE

  - For an expression that converts a non-character type object to a character type, the result character set and collation are **character_set_connection** and **collation_connection**, respectively.
  - By default, the character set and collation number of a binding parameter of the character type are the values of **character_set_connection** and **collation_connection**, respectively. Any value entered for a binding parameter is considered belonging to the preceding character set.
  - During character data conversion, the character encoding is verified. If the character encoding does not meet the requirements, an exception is thrown.

7.4.2 Database-level Character Sets and Collations

When creating a database, you can specify the character set and collation of the database.

```
CREATE DATABASE [IF NOT EXISTS] database_name
        [ ENCODING [=] encoding ] |
        [ LC_COLLATE [=] lc_collate ] |
        [ LC_CTYPE [=] lc_ctype ] |;
```

Syntax description:

- **database_name**

  Specifies the database name.

  Value range: a string. It must comply with the identifier naming convention.

- **ENCODING [=] encoding**

Specifies the character encoding used by the database. The value can be a string (for example, **'SQL_ASCII'**) or an integer.

- **LC_COLLATE [=] 'lc_collate'**

  Specifies the character set used by the new database. For example, set this parameter by using **lc_collate = 'zh_CN.gbk'**.

  The use of this parameter affects the sort order of strings (for example, the order of using ORDER BY for execution and the order of using indexes on text columns). By default, the sorting order of the template database is used.

  Value range: character sets supported by the OS.

- **LC_CTYPE [=] 'lc_ctype'**

  Specifies the character class used by the new database. For example, set this parameter by using **lc_ctype = 'zh_CN.gbk'**. The use of this parameter affects the character class, such as uppercase letters, lowercase letters, and digits. By default, the character class of the template database is used.

  Value range: character classes supported by the OS.

  NOTE

  - The database-level character set and collation syntax can be used in all modes. For details about the syntax, see **CREATE DATABASE**.
  - The LC_COLLATE/LC_CTYPE syntax does not support the collations specific to B-compatible mode. The parameter value range depends on the character sets supported by the local environment. You can run the **locale -a** command to view the parameter value range.

7.4.3 Schema-level Character Sets and Collations

Create a schema and specify the default character set and collation.

```
CREATE SCHEMA schema_name
   [ [DEFAULT] CHARACTER SET | CHARSET [ = ] default_charset ]
   [ [DEFAULT] COLLATE [ = ] default_collation ];
```

Modify the default character set and collation attributes of the schema.

```
ALTER SCHEMA schema_name
   [ [DEFAULT] CHARACTER SET | CHARSET [ = ] default_charset ] [ [DEFAULT] COLLATE [ = ]
default_collation ];
```

Syntax description:

- **schema_name**

  Specifies the schema name.

  Value range: a string. It must comply with the identifier naming convention.

- **default_charset**

  Specifies the default character set of a schema. If you specify a character set separately, the default collation of the schema is set to the default collation of the specified character set.

- **default_collation**

  Specifies the default collation of a schema. If you specify a collation separately, the default character set of the schema is set to the character set corresponding to the specified collation.

GaussDB selects a character set and collation of a schema in the following ways:

- If both **default_charset** and **default_collation** are set, the character set **default_charset** and collation **default_collation** are used. In addition, **default_charset** and **default_collation** must correspond to each other. Otherwise, an error is reported.
- If only **default_charset** is set, the character set **default_charset** and its default collation are used.
- If only **default_collation** is set, the collation **default_collation** and its corresponding character set are used.
- If neither **default_charset** nor **default_collation** is specified, the schema has no default character set or default collation.

NOTE

- Only character sets with a default collation support **default_charset**. If the specified character set does not have a default collation, an error is reported.
- Only the collations in B-compatible mode (**sql_compatibility** is set to **'B'**) support **default_collation**. If another collation is specified, an error is reported.
- The character set of the new schema must be server_encoding of the database.

Example:

```
-- Set only the character set. The collation is the default collation of the character set.
gaussdb=# CREATE SCHEMA test CHARSET utf8;

-- Set only the collation. The character set is the character set associated with the collation.
gaussdb=# CREATE SCHEMA test COLLATE utf8_bin;

-- Set both the character set and collation. The character set and collation must correspond to each other.
gaussdb=# CREATE SCHEMA test CHARSET utf8 COLLATE utf8_bin;

-- Change the default character set of test to utf8mb4 and the default collation to utf8mb4_bin.
gaussdb=# ALTER SCHEMA test CHARSET utf8mb4 COLLATE utf8mb4_bin;
```

7.4.4 Table-level Character Sets and Collations

Set the default character set and default collation for a table.

```
CREATE TABLE table_name (column_list)
    [ [DEFAULT] CHARACTER SET | CHARSET [ = ] default_charset ]
    [ [DEFAULT] COLLATE [ = ] default_collation ]
```

Modify the default character set and collation of a table. The modification does not affect the existing columns in the table.

```
ALTER TABLE table_name
    [ [DEFAULT] CHARACTER SET | CHARSET [ = ] default_charset ]
    [ [DEFAULT] COLLATE [ = ] default_collation ]
```

Modify the default character set and collation of a table to the specified values, set the character set and collation of all columns with character type to the specified value, and convert the data in the column to new character set encoding.

```
ALTER TABLE table_name
    CONVERT TO CHARACTER SET | CHARSET charset [ COLLATE collation ]
```

Parameters

- **table_name**

  Specifies the name of the table.
- **default_charset**

Specifies the default character set of a table. If you specify a character set separately, the default collation of the table is set to the default collation of the specified character set.

- **default_collation**

  Specifies the default collation of a table. If you specify a collation separately, the default character set of the table is set to the character set corresponding to the specified collation.

GaussDB selects a character set and collation of a table in the following ways:

- If both **default_charset** and **default_collation** are set, the character set **default_charset** and collation **default_collation** are used. In addition, **default_charset** and **default_collation** must correspond to each other. Otherwise, an error is reported.
- If only **default_charset** is set, the character set **default_charset** and its default collation are used.
- If only **default_collation** is set, the collation **default_collation** and its corresponding character set are used.
- If neither **default_charset** nor **default_collation** is set, the default character set and collation of the schema where the table is located are used as the default character set and collation of the table.

NOTE

- Only character sets with a default collation support **default_charset**. If the specified character set does not have a default collation, an error is reported.
- Only the collations in B-compatible mode (**sql_compatibility = 'B'**) support **default_collation**. If another collation is specified, an error is reported.
- If the default collation of a table is binary, the text types whose collation is not specified in the table are converted to the corresponding binary type, and the collation is set to binary.
- Currently, the default character set of a table must be server_encoding of the database.

Example:

```
-- Set only the character set. The collation is the default collation of the character set.
gaussdb=# CREATE TABLE test(c1 text) CHARSET utf8;

-- Set only the collation. The character set is the character set associated with the collation.
gaussdb=# CREATE TABLE test(c1 text) COLLATE utf8_bin;

-- Set both the character set and collation. The character set and collation must correspond to each other.
gaussdb=# CREATE TABLE test(c1 text)  CHARSET utf8 COLLATE utf8_bin;

-- Convert the column data of the character type in the table to utf8mb4, and set the collation of the table and column to utf8mb4_bin.
gaussdb=# ALTER TABLE test CONVERT TO CHARSET utf8mb4 COLLATE utf8mb4_bin;

-- Change the default character set of the table to utf8mb4 and the default collation to utf8mb4_bin.
gaussdb=# ALTER TABLE test CHARSET utf8mb4 COLLATE utf8mb4_bin;
```

7.4.5 Column-level Character Sets and Collations

You can set the character set and collation for each column of the string type (CHAR, VARCHAR, or TEXT).

```
CREATE TABLE table_name (
    column_name data_type
    [ CHARACTER SET | CHARSET charset ]
```



```
    [ COLLATE collation ]
);
```

Syntax description:

- **table_name**

  Specifies the name of the table.

- **data_type**

  Specifies the data type of a column. The value can be character set or collation syntax.

- **CHARACTER SET | CHARSET charset**

  Specifies the character set of a table column. If this parameter is specified separately, the collation of the table column is set to the default collation of the specified character set.

- **COLLATE collation**

  The COLLATE clause specifies the collation of a column (the data type of the column must support collation). If no collation is specified, the default collation is used.

GaussDB selects a character set and collation of a table column in the following ways:

- If both **charset** and **collation** are specified, **charset** and **collation** are used. **charset** and **collation** must correspond to each other. Otherwise, an error is reported.
- If only **charset** is specified, the character set **charset** and its default collation are used.
- If only **collation** is specified, the character set associated with collation and the specified collation are used.
- If neither **charset** nor **collation** is specified, the default character set and collation of the table are used.

NOTE

- Only character sets with a default collation support **default_charset**. If the specified character set does not have a default collation, an error is reported.
- Only the collations in B-compatible mode support **default_collation**. If another collation is specified, an error is reported.
- If a table column is of the text type and the specified collation is binary, the text type is converted to the corresponding binary type and the collation is the specified binary collation.
- The character set of the partition key of the partitioned table must be the same as that of the database.
- Currently, the default character set of a table column must be server_encoding of the database.

Example:

```
-- Set only the character set. The collation is the default collation of the character set.
gaussdb=# CREATE TABLE test(c1 text CHARSET utf8);

-- Set only the collation. The character set is the character set associated with the collation.
gaussdb=# CREATE TABLE test(c1 text COLLATE utf8_bin);

-- Set both the character set and collation. The character set and collation must correspond to each other.
gaussdb=# CREATE TABLE test(c1 text CHARSET utf8 COLLATE utf8_bin);
```


7.4.6 Character Sets and Collations of Expressions of the String Type

Each expression of the string type contains character sets and collation attributes.

In B-compatible mode (**sql_compatibility** is set to **'B'**), if **b_format_version** is set to **'5.7'** and **b_format_dev_version** is set to **'s2'**, the default character sets and collations of string constants are determined by the system parameters **character_set_connection** and **collation_connection**. Otherwise, the default character set is the same as server_encoding of the database, and the default collation is **default**.

Character set syntax:

Currently, the character set syntax for specifying string constants is not supported in GaussDB.

```
[_charset_name]'string'
```

Collation syntax:

You can also specify the collation for expressions of other character string types.

```
expression [COLLATE collation_name]
```

Syntax description:

COLLATE collation_name

Collation name, which is the collation attribute of the string.

The data type of the expression must be a data type that supports collations.

The specified collation must be supported by the character set of the expression.

Example:

```
-- Use the COLLATE statement to specify the collation.
gaussdb=# SELECT 'a' COLLATE utf8mb4_general_ci = 'A';
```

7.4.7 Rules for Combining Character Sets and Collations

In B-compatible database (**sql_compatibility = 'B'**), if **b_format_version** is set to **'5.7'** and **b_format_dev_version** is set to **'s2'**, expressions with different character sets and collations are processed based on certain priorities to determine the collations used for character string comparison and the character sets of the expressions.

Collation priority

The priorities of different expressions in descending order are as follows:

- The COLLATE syntax has the highest priority.
- Expressions with collation conflicts (for example, two character strings with different collations).
- Columns of data types that support collation, user-defined variables, stored procedure parameters, and CASE expressions.
- Specific system functions (such as version() and opengauss_version() function expressions).

- String constants and bound parameters.
- NULL expression.
- If a data type of an expression does not support collation, this expression has the lowest priority.

When the collations of two expressions are different, the collation of the expression with the highest priority is used.

Example:

```
gaussdb=#  CREATE TABLE t_utf8(c1 varchar(16) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin);
gaussdb=# INSERT INTO t_utf8 VALUES('STRING');

-- If the utf8mb4_bin collation is used for comparison, the result is false.
gaussdb=# SELECT c1 = 'string' AS result FROM t_utf8;
 result
--------
 f
(1 row)

-- If the utf8mb4_general_ci collation is used for comparison, the result is true.
gaussdb=# SELECT c1 = 'string' COLLATE utf8mb4_general_ci AS result FROM t_utf8;
 result
--------
 t
(1 row)

-- Define the collation of the bound parameter $1 as collation_connection.
gaussdb=# PREPARE test_collation(text) AS SELECT c1 = $1 AS result FROM t_utf8;

-- The collation of the bound parameter is at the same level as that of the string constant. Even if the input expression contains an explicit collation, the collation of c1 is still used for comparison.
gaussdb=# EXECUTE test_collation('string' COLLATE utf8mb4_general_ci);
 result
--------
 f
(1 row)

-- The CASE expression is at the same level as the c1 column. Even if the expression contains an explicit collation, the collation of the c1 column is still used for comparison. The two collations are different. "same level" is displayed.
gaussdb=# SELECT CASE 'string' COLLATE utf8mb4_general_ci WHEN c1 THEN 'different level' ELSE 'same level' END AS result FROM t_utf8;
   result
------------
 same level
(1 row)

-- The IN subquery has the same level as the c1 column. Even if the expression contains an explicit collation, the collation of c1 is still used for comparison. The two collations are different.
gaussdb=# SELECT c1 FROM t_utf8 WHERE c1 in (SELECT 'string' COLLATE utf8mb4_general_ci);
 c1
----
(0 rows)
```

If the collations of two expressions with the same priority are different, the following processing method is used:

- If the two character sets are the same, the collation suffixed with **_bin** is preferred.
- If the two character sets are the same, the default collation is not preferred.
- If the preceding conditions are not met, the two expressions are marked as a collation conflict, and the collations are marked as invalid.
  - If a conflict occurs because the COLLATE syntax specifies different collations for the same character set, an exception is displayed.

- If the two conflicting collations are supported in B-compatible mode (**sql_compatibility** is set to **'B'**), an exception occurs.
- If an invalid collation is used for sorting operations (such as > and <), an exception occurs.

  During equivalent comparison of character strings, if the collation is invalid, the character strings are compared as binary values.

Example:

```
gaussdb=# CREATE TABLE t_utf8mb4_charset(
    c_utf8_bin varchar(16) character set utf8mb4 collate utf8mb4_bin,
    c_utf8_uni varchar(16) character set utf8mb4 collate utf8mb4_unicode_ci,
    c_utf8_gen varchar(16) character set utf8mb4 collate utf8mb4_general_ci);
gaussdb=# INSERT INTO t_utf8mb4_charset VALUES('STRING', 'String', 'string');

-- The utf8mb4_bin collation is preferentially used for comparison. The result is false.
gaussdb=# SELECT c_utf8_bin = c_utf8_uni FROM t_utf8mb4_charset;

-- Collation conflict. Binary comparison is performed, and the result is false.
gaussdb=# SELECT c_utf8_uni = c_utf8_gen FROM t_utf8mb4_charset;

-- Conflict of the explicitly specified collation. An exception is reported.
gaussdb=# SELECT c_utf8_uni COLLATE utf8mb4_unicode_ci = c_utf8_gen COLLATE utf8mb4_general_ci
FROM t_utf8mb4_charset;
```

NOTE

- Only the character sets of objects and expressions of the string type (excluding "char", name, and clob) can be different from those of the database.
- Data types such as ARRAY, XML, JSON, and TSVECTOR contain text data. The character sets in the text data of the objects and expressions of these data types must be the character sets of the database.
- According to the rules for combining character sets and collations, the character sets corresponding to C, POSIX, and DEFAULT collations are server_encoding.

7.5 Constant and Macro

Table 7-26 lists the constants and macros that can be used in GaussDB.

Table 7-26 Constant and macro

Parameter	Description	Example
CURRENT_CATALOG	Specifies the current database.	testdb=# SELECT CURRENT_CATALOG; current_database ------------------ testdb (1 row)
CURRENT_ROLE	Specifies the current user.	gaussdb=# SELECT CURRENT_ROLE; current_user -------------- omm (1 row)
CURRENT_SCHEMA	Specifies the current database mode.	gaussdb=# SELECT CURRENT_SCHEMA; current_schema ---------------- public (1 row)

Parameter	Description	Example
CURRENT_USER	Specifies the current user.	gaussdb=# SELECT CURRENT_USER; current_user -------------- omm (1 row)
LOCALTIMESTAMP	Specifies the current session time (without time zone).	gaussdb=# SELECT LOCALTIMESTAMP; timestamp ---------- 2015-10-10 15:37:30.968538 (1 row)
NULL	This parameter is left blank.	N/A
SESSION_USER	Specifies the current system user.	gaussdb=# SELECT SESSION_USER; session_user -------------- omm (1 row)
SYSDATE	Specifies the current system date.	gaussdb=# SELECT SYSDATE; sysdate ---------- 2015-10-10 15:48:53 (1 row)
USER	Specifies the current user, also called **CURRENT_USER**.	gaussdb=# SELECT USER; current_user -------------- omm (1 row)

7.6 Functions and Operators

Operators can be used to process one or more operands and can be placed before, after, or between operands. Results are returned after the processing.

Functions encapsulate service logic to implement specific functions. A function may or may not have parameters. After a function is executed, the result is returned.

Users can modify system functions. However, after the modification, the meaning of the functions may change, which results in disorder in system control. Therefore, users are not allowed to manually modify system functions.

NOTE

- When the GUC parameter **behavior_compat_options** contains the **'enable_funcname_with_argsname'** option, the projection alias displays the complete function.
- When the GUC parameter **enable_volatile_match_index** is set to **ON** and **DBCOMPATIBILITY** is set to **A**, volatile functions can match indexes. The volatile functions may not hit partial indexes. If there are implicit conversions during function execution, the volatile functions may not hit indexes. In the scenario where indexes cannot be hit, if this option is enabled, the volatile functions still cannot hit the indexes.
- When the GUC parameter **enable_immutable_optimization** is set to **ON** and **DBCOMPATIBILITY** is set to **A**, if the parameter of the immutable stored procedure is a constant or an expression that can be converted into a constant (for example, it can be an immutable function, but cannot be a stable or volatile function), the immutable stored procedure is not executed once per row. The immutable stored procedure is executed once for each row when the input parameter is a row expression. In some scenarios, the immutable stored procedure is executed fewer times, but not just once.

7.6.1 Logical Operators

The usual logical operators include AND, OR, and NOT. SQL uses a three-valued logical system with true, false, and null, which represents "unknown". Their priorities are NOT > AND > OR.

Table 7-27 lists the calculation rules, where a and b represent logical expressions.

Table 7-27 Operation rules

a	b	a AND b Result	a OR b Result	NOT a Result
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
TRUE	NULL	NULL	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE
FALSE	NULL	FALSE	NULL	TRUE
NULL	NULL	NULL	NULL	NULL

NOTE

- The operators AND and OR are commutative, that is, you can switch the left and right operand without affecting the result.
- Operations on XML data are not supported.

7.6.2 Comparison Operators

Comparison operators are available for the most data types and return Boolean values.

All comparison operators are binary operators. Only data types that are the same or can be implicitly converted can be compared using comparison operators.

Table 7-28 describes comparison operators provided by GaussDB.

Table 7-28 Comparison operators

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
=	Equal to
<>, !=, or ^=	Not equal to

- For comparison operators <=, <>, >=, and ^=, if there is a space between two symbols, it does not affect normal operations. For !=, if there is a space between the exclamation mark (!) and an equal sign (=), the exclamation mark will be identified as factorial, which may cause the result to be inconsistent with the expected result.
- Comparison operators are available for all relevant data types. All comparison operators are binary operators that returned values of Boolean type. The calculation priority of the inequality sign is higher than that of the equality sign. If the entered data type is different and cannot be implicitly converted, the comparison fails. For example, an expression such as **1 < 2 < 3** is invalid because the less-than sign (<) cannot be used to compare Boolean values and 3.
- Besides, each comparison operator has a corresponding function in the pg_proc system catalog. If the value of the **proleakproof** attribute of the corresponding function is **f**, the function is not used to prevent data leakage. If a user only has the permission for a system view, but does not have the permission for the corresponding table, the query plan may not be optimal when the user searches the system view.
- This operator does not support data of the XML type.

7.6.3 Character Processing Functions and Operators

String functions and operators provided by GaussDB are for concatenating strings with each other, concatenating strings with non-strings, and matching the patterns of strings. Note: Except length-related functions, other functions and operators of string processing functions do not support parameters greater than 1 GB.

- bin(number)

  Description: Returns a binary string of a given number.

Parameter	Type	Description
number	• Integer types: tinyint, smallint, mediumint, int, and bigint. • Unsigned integer types: tinyint unsigned, smallint unsigned, int unsigned, and bigint unsigned. • Character and text types: char, varchar, tinytext, text, mediumtext, and longtext. Only numeric integer strings are supported, and the integer range is within the bigint range. • Floating-point types: float, real, and double. • Fixed-point types: numeric, decimal, and dec. • Boolean type: bool.	Given number.

Return type: text.

Example:

```
gaussdb=# SELECT bin(5);
  bin
--------
  101
(1 row)
```

NOTE

- The bin function takes effect only when **sql_compatibility** is set to **'B'**.
- If the value of the input parameter **number** is a decimal, round it down.
- If the absolute value of the input parameter **number** exceeds the maximum value of the bigint unsigned type, convert the input parameter **number** to the maximum value of the bigint unsigned type.

- bit_length(string)

  Description: Specifies the number of bits occupied by a string.

  Return type: int.

  Example:


```
gaussdb=# SELECT bit_length('world');
 bit_length
------------
         40
(1 row)
```

- btrim(string text [, characters text])

  Description: Removes the longest string consisting only of characters in **characters** (a space by default) from the start and end of **string**.

  Return type: text.

  Example:

```
gaussdb=# SELECT btrim('sring' , 'ing');
 btrim
-------
 sr
(1 row)
```

- char_length(string) or character_length(string)

  Description: Specifies the number of characters in a string.

  Return type: int.

  Example:

```
gaussdb=# SELECT char_length('hello');
 char_length
-------------
           5
(1 row)
```

- dump(expr[, return_fmt [, start_position [, length]]])

  Description: Returns the data type code, byte length, and internal representation of the input expression. **return_fmt** specifies the number system of the internal representation, **start_position** specifies the byte from which the internal representation starts, and **length** indicates the length of the data to be read.

  Return type: text.

  NOTE

  This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an A-compatible database.

- elt(pos,str1,str2,...)

  Description: Returns the *pos*th string.

Parameter	Type	Description
pos	• Integer types: tinyint, smallint, mediumint, int, and bigint. • Unsigned integer types: tinyint unsigned, smallint unsigned, and int unsigned. • Character and text types: char, varchar, tinytext, text, mediumtext, and longtext. Only numeric integer strings are supported, and the integer range is within the bigint range. • Floating-point types: float, real, and double. • Fixed-point types: numeric, decimal, and dec. • Boolean type: bool.	Specified position of the parameter.

Parameter	Type	Description
str1,str2,...	• Integer types: tinyint, smallint, mediumint, int, and bigint. • Unsigned integer types: tinyint unsigned, smallint unsigned, int unsigned, and bigint unsigned. • Character and text types: char, varchar, tinytext, text, mediumtext, and longtext. • Floating-point types: float, real, and double. • Fixed-point types: numeric, decimal, and dec. • Boolean type: bool. • Large object types: tinyblob, blob, mediumblob, and longblob. • Date types: datetime, timestamp, date, and time.	List of character strings.

Return type: text.

Example:

```
gaussdb=# SELECT elt(3, 'a', 'b', 'c');
 elt
-------
   c
(1 row)
```

NOTE

- The elt function takes effect only when **sql_compatibility** is set to **'B'**.
- If the input parameter **pos** is less than 1 or exceeds the number of parameters, **NULL** is returned.

- field(str,str1,str2,str3,...)

  Description: The field function returns the position of **str** in the *{str1,str2,str3,...}* list. The position increases from 1. If **0** is returned, **str** is not found. If **str** is **NULL**, **0** is returned. If the input parameters of a function are

all digits, the comparison is performed based on digits. If the input parameters are all non-digits, the comparison is performed based on character strings. If the input parameters contain both digits and non-digits, the comparison is performed based on the double type.

Parameter	Type	Description
All parameters	• Integer types: tinyint, smallint, mediumint, int, bigint, tinyint unsigned, smallint unsigned, int unsigned, and bigint unsigned. • Boolean types: true and false. • Floating-point types: float(p), float, real, double, float(m, d), double(m, d), and real(m, d). • Fixed-point types: numeric, decimal, and dec. • Text types: tinytext, text, mediumtext, and longtext. • Character string types: char and varchar. • Large object types: tinyblob, blob, mediumblob, and longblob. • Date types: datetime, timestamp, date, and time.	List of character strings.

Return type: int.

Example:

```
gaussdb=# SELECT field( 'abc','1',1,'abc','abcd' );
 field
-------
     3
(1 row)
```


NOTE

- The field function takes effect only when **sql_compatibility** is set to **'B'**.
- Since the version whose **b_format_version** is set to **'5.7'** and **b_format_dev_version** is set to **'s1'**, the SQL_MODE parameter **pad_char_to_full_length** specifies whether to add spaces at the end of the char type, which affects the field comparison result. For details, see **Table 7-8**.
- Since the version whose **b_format_version** is set to **'5.7'** and **b_format_dev_version** is set to **'s1'**, the behavior of the character, binary, numeric, and date and time types is M-compatible, which affects the field comparison result. For details, see **Data Types**. For the floating-point type in the numeric type, the precision may be different from that in MySQL due to different connection parameter settings. Therefore, this scenario is not recommended, or the numeric type is used instead. For details, see **Connection Parameters**.
- Since the version whose **b_format_version** is set to **'5.7'** and **b_format_dev_version** is set to **'s2'**, characters can be escaped and constant character strings can obtain collations. The collations affect the field comparison result. For details, see the SET NAMES syntax in **SET**. For details about the rules for combining different collations of the character types, see **Rules for Combining Character Sets and Collations**.

- insert(str1, pos, len, str2)

  Description: Returns the processing result of *str1*. The substring starts from *pos* and the number of *len* characters in the character string is replaced with *str2*. If any input parameter is **NULL**, the return value is **NULL**.

Parameter	Type	Description
pos, len	• Integer types: tinyint, smallint, mediumint, int, and bigint. • Unsigned integer types: tinyint unsigned, smallint unsigned, int unsigned, and bigint unsigned. • Character and text types: char, varchar, tinytext, text, mediumtext, and longtext. Only numeric integer strings are supported. • Floating-point types: float, real, and double. • Fixed-point types: numeric, decimal, and dec.	**pos** indicates the parameter in the specified position, and **len** indicates the replacement length.

Parameter	Type	Description
str1,str2	• **NULL** • Integer types: tinyint, smallint, mediumint, int, bigint, tinyint unsigned, smallint unsigned, int unsigned, and bigint unsigned. • Floating-point and fixed-point types: float, real, and double; numeric, decimal, and dec. • Character string types: char and varchar. • Text types: tinytext, text, mediumtext, and longtext. • Large object types: tinyblob, blob, mediumblob, and longblob. • Date types: datetime, timestamp, date, and time.	String.

Return value type: The return value of the function is of the text type (both **s1** and **s2** are of the text type) or bytea type (any of **s1** or **s2** is of the bytea type).

Example:

```
gaussdb=# SELECT INSERT('abcdef',2,3,'gg');
 insert
--------
 aggef
(1 row)
```

NOTE

- The insert function takes effect only when **sql_compatibility** is set to **'B'**.
- The range of input parameters of the Int64 type is from –9223372036854775808 to +9223372036854775807. If a value is out of range, an error is reported. M* does not limit the range of input parameters of the numeric type. If an exception occurs, an alarm is generated, indicating that the value is set to the upper or lower limit. The maximum length of the input parameter of the text type is 2^30 – 5 bytes, and the maximum length of the input parameter of the bytea type is 2^30 – 512 bytes.

- instr(text,text,int,int)

  Description: **instr(string1,string2,int1,int2)** returns the position matching *string2* in *string1* for the *int2*th time from the position specified by *int1*. *int1* indicates the start position for matching, and *int2* indicates the number of matching times.

  Return type: int.

  Example:

  ```
  gaussdb=# SELECT instr( 'abcdabcdabcd', 'bcd', 2, 2 );
   instr
  -------
      6
  (1 row)
  ```

- instrb(text,text,int,int)

  Description: **instrb(string1,string2,int1,int2)** returns the position matching *string2* in *string1* for the *int2*th time from the position specified by *int1*. *int1* indicates the start position for matching, and *int2* indicates the number of matching times. Different from the instr function, instrb is calculated in bytes and is not affected by the character set in use.

  Return type: int.

  Example:

  ```
  gaussdb=# SELECT instrb( 'abcdabcdabcd', 'bcd', 2, 2 );
   instrb
  -------
      6
  (1 row)
  ```

  NOTE

  - This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s1** in an A-compatible database.
  - If the values of *int1* and *int2* are decimals, the values are truncated instead of being rounded off.

- lengthb(text/bpchar)

  Description: Obtains the number of bytes of a specified string.

  Return type: int.

  Example:

  ```
  gaussdb=# SELECT lengthb('hello');
   lengthb
  ---------
         5
  (1 row)
  ```

- left(str text, n int)

  Description: Returns the first *n* characters in a string. When *n* is negative, all but the last */n/* characters are returned.

  Return type: text.

  Example:

  ```
  gaussdb=# SELECT left('abcde', 2);
   left
  ------
   ab
  (1 row)
  ```

- length(string bytea, encoding name)

Description: Specifies the number of characters in **string** in the given **encoding**. The **string** must be valid in this encoding.

Return type: int.

Example:

```
gaussdb=# SELECT length('jose', 'UTF8');
 length
--------
      4
(1 row)
```

NOTE

If the length of the bytea type is queried and UTF8 encoding is specified, the maximum length can only be **536870888**.

- locate(substr, str[, pos])

  Description: Returns the position where *substr* appears for the first time in *str*, starting from *pos* (**1** by default). If *substr* cannot be found in *str*, this function returns **0**.

Parameter	Type	Description
substr,str	• Integer types: tinyint, smallint, mediumint, int, bigint, tinyint unsigned, smallint unsigned, int unsigned, and bigint unsigned. • Floating-point types: float and double. • Arbitrary precision type: numeric. • Character types: char, varchar, and text. • Binary types: bytea and blob. • Date/Time types: date, time, datetime, and timestamp.	• **substr** (mandatory): substring to be searched for. • **str** (mandatory): character string to be searched for.
pos	• Integer types: tinyint, smallint, mediumint, int, tinyint unsigned, smallint unsigned, int unsigned, and bigint unsigned. • Floating-point types: float and double. • Arbitrary precision type: numeric. • Character types: char, varchar, and text. • Boolean types: true and false.	(Optional) Start position of the search.

  Return type: int.

  Example:

```
gaussdb=# SELECT locate( 'b','abcabc');
 locate
-------
      2
(1 row)

gaussdb=# SELECT locate( 'b','abcabc',3);
```



```
 locate
--------
    5
(1 row)
```

NOTE

The locate function takes effect only when **sql_compatibility** is set to **'B'**.

- lpad_s(string text, length int [, fill text])

  Description: Fills up **string** to **length** by appending the characters **fill** (a space by default). If the length of **string** is longer than **length**, an error is reported.

  Return type: text.

  NOTE

  In the scenario where this function is in an A-compatible database, the value of **a_format_version** is **10c**, and that of **a_format_dev_version** is **s1**:

  - The **length** parameter indicates the display length of a character string. The display length of a single character is processed based on A-compatible requirements.
  - During the function execution, if the remaining length is 1 and the next character is of the full-width type (2 bytes), a space character is added to the left of the string.
  - If the value of **length** is a decimal, the value is truncated instead of being rounded off.
  - The **string** and **fill** parameters do not comply with the encoding specifications.

  In other cases:

  - The **length** parameter indicates the total length of characters in a character string. The length of a single character is fixed to 1.
  - If the value of **length** is a decimal, the value is rounded off.
  - The **string** and **fill** parameters do not comply with the encoding specifications.

- lpad(string text, length int [, fill text])

  Description: Fills up **string** to **length** by appending the characters **fill** (a space by default). If **string** is already longer than **length**, then it is truncated.

  Return type: text.

NOTE

In the scenario where this function is in an A-compatible database, the value of **a_format_version** is **10c**, and that of **a_format_dev_version** is **s1**:

- The **length** parameter indicates the display length of a character string. The display length of a single character is processed based on O-compatible requirements.
- During the function execution, if the remaining length is 1 and the next character is of the full-width type (2 bytes), a space character is added to the left of the string.
- If the value of **length** is a decimal, the value is truncated instead of being rounded off.
- The **string** and **fill** parameters do not comply with the encoding specifications.

In other cases:

- The **length** parameter indicates the total length of characters in a character string. The length of a single character is fixed to 1.
- If the value of **length** is a decimal, the value is rounded off.
- The **string** and **fill** parameters do not comply with the encoding specifications.

- make_set(bits,str1,str2...)

  Description: **str1** corresponds to bit 0 of the bits input parameter in the bit format, **str2** corresponds to bit 1, and so on. If the lower bit of the corresponding bit form is **1**, the corresponding **str** parameter is appended to the result and returned with a comma as the separator.

Parameter	Type	Description
make_set	The first input parameter **bits** supports the following types: • NULL. • Integer types: tinyint, smallint, mediumint, int, bigint, tinyint unsigned, smallint unsigned, int unsigned, and bigint unsigned. • Floating-point and fixed-point types: float, real, and double; numeric, decimal, and dec. • Bit type: bit. • Character string types: char and varchar. • Text types: tinytext, text, mediumtext, and longtext. • Large object types: tinyblob, blob, mediumblob, and longblob. The second input parameter **str** supports the following types: • NULL. • Integer types: tinyint, smallint, mediumint, int, bigint, tinyint unsigned, smallint unsigned, int unsigned, and bigint unsigned. • Floating-point and fixed-point types: float, real, and double; numeric, decimal, and dec. • Character string types: char and varchar. • Text types: tinytext, text, mediumtext, and longtext. • Large object types: tinyblob, blob, mediumblob, and longblob. • Date types: datetime, timestamp, date, and time.	The returned result consists of some character strings selected from *str1,str2,...,strN* and separated by commas (,).

Return type: text.

Example:

```
gaussdb=#  SELECT make_set(7,'a','b','c','d');
 make_set
----------
 a,b,c
(1 row)
gaussdb=#  SELECT
```



```
make_set(b'10101010101011010101010101101010101010110101010101011010101010101','1','2','3','4','
5','6','7','8','9','10','11','12','13','14','15','16','17','18','19','20','21','22','23','24','25','26','27','28','29','30','31','3
2','33','34','35','36','37','38','39','40','41','42','43','44','45','46','47','48','49','50','51','52','53','54','55','56','57','
58','59','60','61','62','63','64','65','66','67','68','69');
                                                    make_set
-----------------------------------------------------------------------------------------------------
 2,4,6,8,10,12,13,15,17,19,21,23,25,26,28,30,32,34,36,38,39,41,43,45,47,49,51,52,54,56,58,60,62,64
(1 row)
```

NOTE

- The make_set function takes effect only when **sql_compatibility** is set to **'B'**.
- When the **bits** parameter is an integer, the maximum range is int128, which is smaller than the B-compatible range.
- When the **bits** parameter is of the date type (datetime, timestamp, date, or time), it is not supported because the conversion from the date type to the integer type is different from that in B-compatible mode, which is not caused by this function.
- GaussDB and B-compatibility are inherently different in the bit and Boolean types, causing different returned results, which is not caused by this function. When the **bits** input parameter is of the Boolean type, and the **str** input parameter is of the bit or Boolean type, they are not supported.

- notlike(x bytea name text, y bytea text)

  Description: Compares **x** and **y** to check whether they are inconsistent.

  Return type: Boolean.

  Example:

```
gaussdb=# SELECT notlike(1,2);
   notlike
--------------
         t
(1 row)
gaussdb=# SELECT notlike(1,1);
   notlike
--------------
         f
(1 row)
```

- octet_length(string)

  Description: Specifies the number of bytes in a string.

  Return type: int.

  Example:

```
gaussdb=# SELECT octet_length('jose');
 octet_length
--------------
            4
(1 row)
```

- overlay(string placing string FROM int [for int])

  Description: Replaces substrings. **FROM int** indicates the start position of the replacement in the first string. **for int** indicates the number of characters replaced in the first string.

  Return type: text.

  Example:

```
gaussdb=# SELECT overlay('hello' placing 'world' from 2 for 3 );
 overlay
---------
 hworldo
(1 row)
```

- position(substring in string)

Description: Specifies the position of a substring. Parameters are case-sensitive.

Return type: int. If the character string does not exist, **0** is returned.

Example:

```
gaussdb=# SELECT position('ing' in 'string');
 position
----------
        4
(1 row)
```

- pg_client_encoding()

  Description: Specifies the current client encoding name.

  Return type: name.

  Example:

  ```
  gaussdb=# SELECT pg_client_encoding();
   pg_client_encoding
  --------------------
   UTF8
  (1 row)
  ```

- quote(str)

  Description: Returns a string enclosed in single quotation marks and adds a backslash (\) before the instances of backslashes (\), single quotation marks ('), ASCII NUL (\0), and Control+Z (\Z). If the parameter is **NULL**, the return value is **NULL** without single quotation marks.

Parameter	Type	Description
quote	• NULL. • Integer types: tinyint, smallint, mediumint, int, bigint, tinyint unsigned, smallint unsigned, int unsigned, and bigint unsigned. • Floating-point and fixed-point types: float, real, and double; numeric, decimal, and dec. • Character string types: char and varchar. • Text types: tinytext, text, mediumtext, and longtext. • Large object types: tinyblob, blob, mediumblob, and longblob. • Date types: datetime, timestamp, date, and time.	Adds quotation marks to the input character string and adds a backslash (\) before the backslash (\) and single quotation mark (').

Return type: text.

Example:

```
gaussdb=# SELECT quote('hello\' world');
    quote
--------------
 'hello\' world'
(1 row)
```


NOTE

1. The quote function takes effect only when **sql_compatibility** is set to **'B'**.
2. The GUC parameters **set standard_conforming_strings** is set to **off**, **set escape_string_warning** is set to **off**, and **set backslash_quote** is set to **on**.
3. If the **str** character string contains \Z, \r, \%, or _, GaussDB does not escape it, which is different from B-compatible mode. The slash followed by digits may also cause differences, for example, "\563". This function difference is the escape character difference between GaussDB and B-compatible mode, which is not caused by this function.
4. The output format of "\b" in the **str** character string is different from that in B-compatible mode. This is an inherent difference between GaussDB and B-compatible mode and is not caused by this function.
5. If the **str** character string contains "\0", GaussDB cannot identify the character because the UTF-8 character set cannot identify the character. As a result, the input fails. This is an inherent difference between GaussDB and B-compatible mode and is not caused by this function.
6. If **str** is of the bit or Boolean type, this type is not supported because it is different in GaussDB and B-compatible mode.
7. GaussDB supports a maximum of 1 GB data transfer. The maximum length of the **str** input parameter is 536870908, and the maximum size of the result string returned by the function is 1 GB.

- quote_ident(string text)

  Description: Returns the given string suitably quoted to be used as an identifier in an SQL statement string (quotation marks are used as required). Quotation marks are added only if necessary (that is, if the string contains non-identifier characters or would be case-folded). Embedded quotation marks are properly doubled.

  Return type: text.

  Example:

  ```
  gaussdb=# SELECT quote_ident('hello world');
   quote_ident
  --------------
   "hello world"
  (1 row)
  ```

- quote_literal(string text)

  Description: Returns the given string suitably quoted to be used as text in an SQL statement string (quotation marks are used as required).

  It supports XML data that is explicitly converted to the character type.

  Return type: text.

  Example:

  ```
  gaussdb=# SELECT quote_literal('hello');
   quote_literal
  ---------------
   'hello'
  (1 row)
  ```

  If a command similar to the following exists, the text will be escaped:

  ```
  gaussdb=# SELECT quote_literal(E'O\'hello');
   quote_literal
  ---------------
   'O''hello'
  (1 row)
  ```

  If a command similar to the following exists, the backslash will be properly doubled.


```
gaussdb=# SELECT quote_literal('O\hello');
 quote_literal
---------------
 E'O\\hello'
(1 row)
```

If the parameter is null, **NULL** is returned. If the parameter may be null, you are advised to use **quote_nullable**.

```
gaussdb=# SELECT quote_literal(NULL);
 quote_literal
---------------

(1 row)
```

- quote_literal(value anyelement)

  Description: Converts the given value to text and then quotes it as a literal.

  Return type: text.

  Example:

  ```
  gaussdb=# SELECT quote_literal(42.5);
   quote_literal
  ---------------
   '42.5'
  (1 row)
  ```

  If a command similar to the following exists, the given value will be escaped.

  ```
  gaussdb=# SELECT quote_literal(E'O\'42.5');
   quote_literal
  ---------------
   '0''42.5'
  (1 row)
  ```

  If a command similar to the following exists, the backslash will be properly doubled.

  ```
  gaussdb=# SELECT quote_literal('O\42.5');
   quote_literal
  ---------------
   E'O\\42.5'
  (1 row)
  ```

- quote_nullable(string text)

  Description: Returns the given string suitably quoted to be used as a string in an SQL statement string (quotation marks are used as required).

  It supports XML data that is explicitly converted to the character type.

  Return type: text.

  Example:

  ```
  gaussdb=# SELECT quote_nullable('hello');
   quote_nullable
  ----------------
   'hello'
  (1 row)
  ```

  If a command similar to the following exists, the text will be escaped:

  ```
  gaussdb=# SELECT quote_nullable(E'O\'hello');
   quote_nullable
  ----------------
   'O''hello'
  (1 row)
  ```

  If a command similar to the following exists, the backslash will be properly doubled.

  ```
  gaussdb=# SELECT quote_nullable('O\hello');
   quote_nullable
  ----------------
  ```



```
 E'O\\hello'
(1 row)
```

If the parameter is null, **NULL** is returned.

```
gaussdb=# SELECT quote_nullable(NULL);
 quote_nullable
----------------
 NULL
(1 row)
```

- quote_nullable(value anyelement)

  Description: Converts the given value to text and then quotes it as a literal.

  Return type: text.

  Example:

  ```
  gaussdb=# SELECT quote_nullable(42.5);
   quote_nullable
  ----------------
   '42.5'
  (1 row)
  ```

  If a command similar to the following exists, the given value will be escaped.

  ```
  gaussdb=# SELECT quote_nullable(E'O\'42.5');
   quote_nullable
  ----------------
   'O''42.5'
  (1 row)
  ```

  If a command similar to the following exists, the backslash will be properly doubled.

  ```
  gaussdb=# SELECT quote_nullable('O\42.5');
   quote_nullable
  ----------------
   E'O\\42.5'
  (1 row)
  ```

  If the parameter is null, **NULL** is returned.

  ```
  gaussdb=# SELECT quote_nullable(NULL);
   quote_nullable
  ----------------
   NULL
  (1 row)
  ```

- space(count)

  Description: Returns a string consisting of a specified number of spaces.

Parameter	Type	Description
count	• Integer types: tinyint, smallint, mediumint, int, and bigint. • Unsigned integer types: tinyint unsigned, smallint unsigned, and int unsigned. • Character and text types: char, varchar, tinytext, text, mediumtext, and longtext. Only numeric integer strings are supported, and the integer range is within the bigint range. • Floating-point types: float, real, and double. • Fixed-point types: numeric, decimal, and dec. • Boolean type: bool.	Number of spaces.

Return type: text.

Example:

```
gaussdb=# SELECT space(5);
 space
--------


(1 row)
```

NOTE

- The space function takes effect only when **sql_compatibility** is set to **'B'**.
- If the input parameter **count** is less than **1073741819** and greater than **0**, a string of *count* spaces is returned. Otherwise, an empty string is returned.

- substring_inner(string [from int] [for int])

  Description: Extracts a substring. **from int** indicates the start position of the truncation. **for int** indicates the number of characters truncated.

  Return type: text.

  Example:

```
gaussdb=# SELECT substring_inner('adcde', 2,3);
 substring_inner
```



```
-----------------
 dcd
(1 row)
```

- substring(string [from int] [for int])

  Description: Extracts a substring. **from int** indicates the start position of the truncation. **for int** indicates the number of characters truncated.

  Return type: text.

  Example:

```
gaussdb=# SELECT substring('Thomas' from 2 for 3);
 substring
-----------
 hom
(1 row)
```

  NOTE

  This function is in the B-compatible database. When the GUC parameter **b_format_version** is set to **5.7** and **b_format_dev_version** is set to **'s1'**, **[from int]** can be a negative number, indicating the sequence number of the character counted from back to front. If this parameter is not set, **[from int]** is a negative number and the result is empty.

- substring(string, pos, len)

  Description: Extracts a substring. **pos** indicates the start position of the truncation. **len** indicates the number of characters truncated.

  The parameters are described as follows.

  Table 7-29 Parameters

Parameter	Type	Description	Value Range
string	text	Character string to be truncated.	-
pos	int	Start position of the character string to be truncated.	The absolute value is less than the string length.
len	int	Length of the character string to be truncated.	The value is greater than 0.

  Return type: text.

  Example:

```
gaussdb=# SELECT substring('substrteststring', -5, 5);
 substring
-----------
 tring
(1 row)
```

  NOTE

  This function is in the B-compatible database. When the GUC parameter **b_format_version** is set to **'5.7'** and **b_format_dev_version** is set to **'s1'**, **pos** can be a negative number, indicating the sequence number of the character counted from back to front. If this parameter is not set, **pos** is a negative number and the result is empty.

- substring_index(str,split,index)

  Description: **str** is a string, **split** is a separator string, and **index** is the position from which the string is split. This function returns all contents on the left or right (left if the value of **index** is positive and right if the value of **index** is negative) of the **str** string that is split by **split** from the position specified by **index**.

  Parameters: See **Table 7-30**.

  Table 7-30 Parameters of substring_index

Parameter	Type	Description
str	text	Character string to be truncated.
split	text	Target character string to be split.
index	int	Position from which the string is split. If the value of **index** is a positive number, all contents on the left of the position are obtained. If the value of **index** is a negative number, all contents on the right are obtained.

  Return type: text.

  Example:

  ```
  gaussdb=#  SELECT substring_index('Test1splitTest2splitTest3splitTest4', 'split', 2);
   substring_index
  -----------------
   Test1splitTest2
  (1 row)

  gaussdb=#  SELECT substring_index('Test1splitTest2splitTest3splitTest4', 'split', -2);
   substring_index
  -----------------
   Test3splitTest4
  (1 row)
  ```

- substring(string from *pattern*)

  Description: Extracts substrings matching the POSIX regular expression. It returns the text that matches the pattern. If no match record is found, a null value is returned.

  Return type: text.

  Example:

  ```
  gaussdb=# SELECT substring('Thomas' from '...$');
   substring
  -----------
   mas
  (1 row)
  gaussdb=# SELECT substring('foobar' from 'o(.)b');
   result
  --------
   o
  (1 row)
  gaussdb=# SELECT substring('foobar' from '(o(.)b)');
   result
  --------
  ```



```
 oob
(1 row)
```

NOTE

If the POSIX regular expression contains any parentheses, the portion of the text that matched the first parenthesized sub-expression (the one whose left parenthesis comes first) is returned. You can put parentheses around the whole expression if you want to use parentheses within it without triggering this exception.

- substring(string from *pattern* for *escape*)

  Description: Extracts substrings matching the SQL regular expression. The declared schema must match the entire data string; otherwise, the function fails and returns a null value. To indicate the part of the schema that should be returned on success, the schema must contain two occurrences of the escape character followed by a double quotation mark ("). The text matching the portion of the pattern between these marks is returned.

  Return type: text.

  Example:

  ```
  gaussdb=# SELECT substring('Thomas' from '%#"o_a#"_' for '#');
   substring
  -----------
   oma
  (1 row)
  ```

- rawcat(raw,raw)

  Description: Indicates the string concatenation function.

  Return type: raw.

  Example:

  ```
  gaussdb=# SELECT rawcat('ab','cd');
   rawcat
  --------
   ABCD
  (1 row)
  ```

- regexp_like(text,text,text)

  Description: Indicates the mode matching function of a regular expression.

  Return type: Boolean.

  Example:

  ```
  gaussdb=# SELECT regexp_like('str','[ac]');
   regexp_like
  -------------
   f
  (1 row)
  ```

- regexp_substr(string text, pattern text [, position int [, occurrence int [, flags text]]])

  Description: Extracts substrings from a regular expression. Its function is similar to **substr**. When a regular expression contains multiple parallel brackets, it also needs to be processed.

  Parameter description:

  – **string**: source character string used for matching.

  – **pattern**: regular expression pattern string used for matching.

  – **position**: start character of the source string used for matching. This parameter is optional. The default value is **1**.

- **occurrence**: sequence number of the matched substring to be extracted. This parameter is optional. The default value is **1**.
- **flags**: contains zero or multiple single-letter flags that change the matching behavior of the function. This parameter is optional. The options supported by **flags** and description are described in **Table 7-31**.

Table 7-31 Options supported by flags

Option	Description
'b'	Indicates the BRE matching without extension.
'c'	Indicates the case-sensitive matching.
'e'	Indicates the ERE matching with extension.
'i'	Indicates the case-insensitive matching.
'm'	Indicates the multi-line matching. If **flags** contains **'m'**, use the multi-line matching. Otherwise, use the single-line matching.
'n'	The meaning of **'n'** is related to the GUC parameter **behavior_compat_options** and the compatibility mode of the current database. • If the SQL compatibility mode of the database is **A** or **B** and the value of the GUC parameter **behavior_compat_options** contains **aformat_regexp_match**, the **n** option indicates that "." matches the linefeed **'\n'**. If **'n'** is not specified, "." does not match the linefeed. • In other cases, the **'n'** option has the same meaning as the **'m'** option.
'p'	Indicates partial linefeed-sensitive matching, which is similar to the linefeed-sensitive matching (**'m'** or **'n'**) and affects "." and square bracket expression, but does not affect ^ and $.
'q'	Indicates common character matching.
's'	Indicates the single-line matching. The meaning is opposite to those of **'m'** and **'n'**.
't'	Indicates the compact matching. The whitespace characters match themselves.
'w'	Indicates the reverse partial linefeed-sensitive matching. The meaning is opposite to that of **'p'**.
'x'	Indicates the loose matching. The whitespace characters are ignored.

Return type: text.

Example:


```
gaussdb=# SELECT regexp_substr('str','[ac]');
 regexp_substr
---------------

(1 row)

gaussdb=# SELECT regexp_substr('foobarbaz', 'b(..)', 3, 2) AS RESULT;
 result
--------
 baz
(1 row)
```

- regexp_count(string text, pattern text [, position int [, flags text]])

  Description: obtains the number of substrings used for matching.

  Parameters:

  - **string**: source character string used for matching.
  - **pattern**: regular expression pattern string used for matching.
  - **position**: start character of the source string used for matching. This parameter is optional. The default value is **1**.
  - **flags**: contains zero or multiple single-letter flags that change the matching behavior of the function. This parameter is optional. The options supported by **flags** and description are described in **Table 7-31**.

  NOTE

  When the function is in an A-compatible database, the value of **a_format_version** is **10c**, and the value of **a_format_dev_version** is **s1**, the **pattern** parameter ending with a backslash (\) is valid.

  Return type: int.

  Example:

```
gaussdb=# SELECT regexp_count('foobarbaz','b(..)', 5) AS RESULT;
result
--------
1
(1 row)
```

- regexp_instr(string text, pattern text [, position int [, occurrence int [, return_opt int [, flags text]]]])

  Description: obtains the position (starting from 1) of the substring that meets the matching condition. If no substring is matched, **0** is returned.

  Parameter description:

  - **string**: source character string used for matching.
  - **pattern**: regular expression pattern string used for matching.
  - **position**: start character of the source string used for matching. This parameter is optional. The default value is **1**.
  - **occurrence**: sequence number of the matched substring to be obtained. This parameter is optional. The default value is **1**.
  - **return_opt**: specifies whether to return the position of the first or last character of the matched substring. This parameter is optional. If the value is **0**, the position of the first character (starting from 1) of the matched substring is returned. If the value is greater than 0, the position of the next character of the end character of the matched substring is returned. The default value is **0**.

  - **flags**: contains zero or multiple single-letter flags that change the matching behavior of the function. This parameter is optional. The options supported by **flags** and description are described in **Table 7-31**.

  Return type: int.

  Example:

```
gaussdb=# SELECT regexp_instr('foobarbaz','b(..)', 1, 1, 0) AS RESULT;
result
--------
4
(1 row)

gaussdb=# SELECT regexp_instr('foobarbaz','b(..)', 1, 2, 0) AS RESULT;
result
--------
7
(1 row)
```

- regexp_matches(string text, pattern text [, flags text])

  Description: Returns all captured substrings resulting from matching a POSIX regular expression against **string**. If the pattern does not match, the function returns no rows. If the pattern contains no parenthesized sub-expressions, then each row returned is a single-element text array containing the substring matching the whole pattern. If the pattern contains parenthesized sub-expressions, the function returns a text array whose *n*th element is the substring matching the *n*th parenthesized sub-expression of the pattern.

  The optional **flags** argument contains zero or multiple single-letter flags that change the function behavior. **i** indicates that the matching is not related to uppercase and lowercase. **g** indicates that each matched substring is replaced, instead of replacing only the first one.

  NOTICE

  If the last parameter is provided but the parameter value is an empty string ('') and the SQL compatibility mode of the database is set to A, the returned result is an empty set. This is because the A compatibility mode treats the empty string ('') as **NULL**. To resolve this problem, you can:

  - Change the database SQL compatibility mode to C.
  - Do not provide the last parameter or do not set the last parameter to an empty string.

  Return type: SETOF text[].

  Example:

```
gaussdb=# SELECT regexp_matches('foobarbequebaz', '(bar)(beque)');
 regexp_matches
----------------
 {bar,beque}
(1 row)
gaussdb=# SELECT regexp_matches('foobarbequebaz', 'barbeque');
 regexp_matches
----------------
 {barbeque}
(1 row)
gaussdb=#  SELECT regexp_matches('foobarbequebazilbarfbonk', '(b[^b]+)(b[^b]+)', 'g');
 regexp_matches
----------------
 {bar,beque}
```



```
 {bazil,barf}
(2 rows)
```

- regexp_match(string text, pattern text [, flags text])

  Description: Returns a string array, which is the first substring obtained by matching the POSIX regular expression with the specified string. If the pattern does not match, the function returns a blank line. If the pattern contains no parenthesized sub-expressions, then each row returned is a single-element text array containing the substring matching the whole pattern. If the pattern contains parenthesized sub-expressions, the function returns a text array whose *n*th element is the substring matching the *n*th parenthesized sub-expression of the pattern.

  The **flags** parameter is optional. If **flags** is not specified, tight syntax matching is used by default. The **flags** parameter contains zero or multiple single-letter flags that change the behavior of a function. **Table 4 Parameters of flags** lists the supported flags.

  Table 7-32 Parameters of flags

Option	Description
'b'	Converts the POSIX regular expression to a BRE.
'c'	Indicates the case-sensitive matching.
'e'	Converts the POSIX regular expression to an ERE.
'i'	Indicates the case-insensitive matching.
'n'	Indicates newline-sensitive matching. A character string is regarded as multiple lines. Caret (^) and dollar sign ($) match the beginning and end of each line. Periods (.) do not match linefeed.
'm'	Indicates newline-sensitive matching. A character string is regarded as multiple lines. Caret (^) and dollar sign ($) match the beginning and end of each line. Periods (.) do not match linefeed. The matching mode of *n* is the same.
'p'	Indicates partial newline-sensitive matching. Periods (.) do not match linefeed. Caret (^) can only match the beginning of the first line and dollar sign ($) can only match the end of the last line.
'q'	Represents the regular expression is a text string, all of which are common characters.
's'	Indicates non-newline-sensitive matching. That is, a character string is regarded as a single line, and caret (^) and dollar sign ($) match the beginning and end of the line, respectively. In this mode, periods (.) in the regular expression can match any character, including the linefeed character.
't'	Indicates the tight syntax matching.

Option	Description
'w'	Indicates inverse partial newline-sensitive matching. In the regular expression, caret (^) matches the beginning of each line, and dollar sign ($) matches only the end of the last line. Period (.) can match any character, including the linefeed character.
'x'	Indicates the extended syntax matching, ignoring whitespace characters and comments in regular expressions, except that: • The whitespace characters or comments starting with the number sign (#) that are prefixed with slashes (\) will be retained. • The whitespace characters or comments starting with the number sign (#) in square brackets will be retained. • The whitespace characters and comments are not allowed in multi-character symbols, such as, left parenthesis ((), question mark (?), and colon (:).

NOTICE

If the parameter contains an empty string ('') and the SQL compatibility mode of the database is set to **A**, the returned result is **NULL**. This is because the empty string is processed as **NULL** in A-compatible mode.

Return type: SETOF text[].

Example:

```
gaussdb=# SELECT regexp_match('foobarbequebaz', '(bar)(beque)');
 regexp_match
--------------
 {bar,beque}
(1 row)
gaussdb=# SELECT (regexp_match('foobarbequebaz', 'bar.*que'))[1];
 regexp_match
--------------
 barbeque
(1 row)
gaussdb=# SELECT regexp_match('Learning #PostgreSQL', 'R', 'c');
 regexp_match
--------------

(1 row)
gaussdb=# SELECT regexp_match('hello world', 'h e l l o', 'x');
 regexp_match
--------------
 {hello}
(1 row)
```

- regexp_split_to_array(string text, pattern text [, flags text])

  Description: Splits **string** using a POSIX regular expression as the delimiter. The regexp_split_to_array function behaves the same as regexp_split_to_table, except that regexp_split_to_array returns its result as an array of text.

  Return type: text[].

  Example:


```
gaussdb=# SELECT regexp_split_to_array('hello world', E'\\s+');
 regexp_split_to_array
-----------------------
 {hello,world}
(1 row)
```

- regexp_split_to_table(string text, pattern text [, flags text])

  Description: Splits **string** using a POSIX regular expression as the delimiter. If there is no match to the pattern, the function returns the string. If there is at least one match, for each match it returns the text from the end of the last match (or the beginning of the string) to the beginning of the match. When there are no more matches, it returns the text from the end of the last match to the end of the string.

  The **flags** parameter is a text string containing zero or more single-letter flags that change the function's behavior. **i** indicates case-insensitive matching.

  Return type: SETOF text.

  Example:

```
gaussdb=# SELECT regexp_split_to_table('hello world', E'\\s+');
 regexp_split_to_table
-----------------------
 hello
 world
(2 rows)
```

- repeat(string text, number int)

  Description: Repeats **string** the specified number of times.

  Return type: text.

  Example:

```
gaussdb=# SELECT repeat('Pg', 4);
  repeat
----------
 PgPgPgPg
(1 row)
```

  NOTE

  The maximum size of memory allocated at a time cannot exceed 1 GB due to the memory allocation mechanism of the database. Therefore, the maximum value of **number** cannot exceed (1 GB – **x**)/**lengthb** (**string**) – 1. **x** indicates the length of the header information, which is usually greater than 4 bytes. The value varies among different scenarios.

- replace(string text, from text, to text)

  Description: Replaces all occurrences in **string** of substring **from** with substring **to**.

  Return type: text.

  Example:

```
gaussdb=# SELECT replace('abcdefabcdef', 'cd', 'XXX');
    replace
----------------
 abXXXefabXXXef
(1 row)
```

- replace(string, substring)

  Description: Deletes all substrings in a string.

  String type: text.

  Substring type: text.

Return type: text.

Example:

```
gaussdb=# SELECT replace('abcdefabcdef', 'cd');
    replace
----------------
 abefabef
(1 row)
```

- reverse(str)

  Description: Returns a reversed string (by character).

  Return type: text.

  Example:

  ```
  gaussdb=# SELECT reverse('abcde');
   reverse
  ---------
   edcba
  (1 row)
  ```

- right(str text, n int)

  Description: Returns the last *n* characters in a string. When *n* is negative, all but the first */n/* characters are returned.

  Return type: text.

  Example:

  ```
  gaussdb=# SELECT right('abcde', 2);
   right
  -------
   de
  (1 row)

  gaussdb=# SELECT right('abcde', -2);
   right
  -------
   cde
  (1 row)
  ```

- rpad(string text, length int [, fill text])

  Description: Fills up **string** to **length** by appending the characters **fill** (a space by default). If **string** is already longer than **length**, then it is truncated.

  Return type: text.

NOTE

In the scenario where this function is in an A-compatible database, the value of **a_format_version** is **10c**, and that of **a_format_dev_version** is **s1**:

- The **length** parameter indicates the display length of a character string. The display length of a single character is processed based on O-compatible requirements.
- During the function execution, if the remaining length is 1 and the next character is of the full-width type (2 bytes), a space character is added to the right of the string.
- If the value of **length** is a decimal, the value is truncated instead of being rounded off.
- The **string** and **fill** parameters do not comply with the encoding specifications.

In other cases:

- The **length** parameter indicates the total length of characters in a character string. The length of a single character is fixed to 1.
- If the value of **length** is a decimal, the value is rounded off.
- The **string** and **fill** parameters do not comply with the encoding specifications.

- substrb(text,int,int)

  Description: Extracts a substring. The first **int** indicates the start position of the subtraction. The second **int** indicates the number of characters extracted.

  Return type: text.

  Example:

```
gaussdb=# SELECT substrb('string',2,3);
 substrb
---------
 tri
(1 row)
```

- substrb(text,int)

  Description: Extracts a substring. **int** indicates the start position of the extraction.

  Return type: text.

  Example:

```
gaussdb=# SELECT substrb('string',2);
 substrb
---------
 tring
(1 row)
```

- substr(text,int)/substr(str FROM pos)

  Description: Outputs the string specified by **str** from the position specified by **pos** to the end of the character string, in which **str** is the target string, and **pos** is a position in the string.

  Return type: text.

  Example:

```
gaussdb=# SELECT substr('stringtest' from 4);
 substr
---------
 ingtest
(1 row)

gaussdb=# SELECT substr('stringtest', 4);
 substr
---------
```



```
 ingtest
(1 row)
```

- substr(str FROM pos FOR len)

  Description: Extracts a substring. The first **int** indicates the start position of the subtraction. The second **int** indicates the number of characters extracted. The value of **pos** can be a negative number. If the value is a negative number, the value is extracted from back to front.

  Return type: text.

  Example:

```
gaussdb=# SELECT substr('teststring' from 5 for 2);
 substr
--------
 st
(1 row)
```

- substr(bytea,from,count)

  Description: Extracts a substring from **bytea**. **from** specifies the position where the extraction starts. **count** specifies the length of the extracted substring.

  Return type: text.

  Example:

```
gaussdb=# SELECT substr('string',2,3);
 substr
--------
 tri
(1 row)
```

- string || string

  Description: Concatenates strings.

  Return type: text.

  Example:

```
gaussdb=# SELECT 'MPP'||'DB' AS RESULT;
 result
--------
 MPPDB
(1 row)
```

- string || non-string or non-string || string

  Description: Concatenates strings and non-strings.

  Return type: text.

  Example:

```
gaussdb=# SELECT 'Value: '||42 AS RESULT;
  result
-----------
 Value: 42
(1 row)
```

- split_part(string text, delimiter text, field int)

  Description: Splits **string** on **delimiter** and returns the *field*th column (counting from text of the first appeared delimiter).

  Return type: text.

  Example:

```
gaussdb=# SELECT split_part('abc~@~def~@~ghi', '~@~', 2);
 split_part
------------
```



```
 def
(1 row)
```

- strpos(string, substring)

  Description: Specifies the position of a substring. It is the same as **position(substring in string)**. However, the parameter sequences of them are reversed.

  Return type: int.

  Example:

```
gaussdb=# SELECT strpos('source', 'rc');
 strpos
--------
      4
(1 row)
```

- to_hex(number int or bigint)

  Description: Converts a number to a hexadecimal expression.

  Return type: text.

  Example:

```
gaussdb=# SELECT to_hex(2147483647);
  to_hex
----------
 7fffffff
(1 row)
```

- translate(string text, from text, to text)

  Description: Any character in **string** that matches a character in **from** is replaced by the corresponding character in **to**. If **from** is longer than **to**, extra characters occurred in **from** are removed.

  Return type: text.

  Example:

```
gaussdb=# SELECT translate('12345', '143', 'ax');
 translate
-----------
 a2x5
(1 row)
```

- length(string)

  Description: Obtains the number of characters in a string. When **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, if the string is of the string or text type, the number of bytes of the string is returned.

  Return type: integer.

- lengthb(string)

  Description: Obtains the number of bytes in a string. The value depends on character sets (GBK and UTF8).

  Return type: integer.

  Example:

```
gaussdb=# SELECT lengthb('Chinese');
 lengthb
---------
       7
(1 row)
```

- substr(string,from)

  Description:

Extracts substrings from a string.

from indicates the start position of the extraction.

- If **from** starts at 0, the value **1** is used.
- If the value of **from** is positive, all characters from **from** to the end are extracted.
- If the value of **from** is negative, the last *n* characters in the string are extracted, and *n* indicates the absolute value of **from**.

Return type: varchar.

Example:

If the value of **from** is positive:

```
gaussdb=# SELECT substr('ABCDEF',2);
 substr
--------
 BCDEF
(1 row)
```

If the value of **from** is negative:

```
gaussdb=# SELECT substr('ABCDEF',-2);
 substr
--------
 EF
(1 row)
```

- substr(string,from,count)

  Description:

  Extracts substrings from a string.

  from indicates the start position of the extraction.

  count indicates the length of the extracted substring.

  - If **from** starts at 0, the value **1** is used.
  - If the value of **from** is positive, extract **count** characters starting from **from**.
  - If the value of **from** is negative, extract the last *n* **count** characters in the string, in which *n* indicates the absolute value of **from**.
  - If the value of **count** is smaller than **1**, **null** is returned.

  Return type: varchar.

  Example:

  If the value of **from** is positive:

  ```
  gaussdb=# SELECT substr('ABCDEF',2,2);
   substr
  --------
   BC
  (1 row)
  ```

  If the value of **from** is negative:

  ```
  gaussdb=# SELECT substr('ABCDEF',-3,2);
   substr
  --------
   DE
  (1 row)
  ```

- substrb(string,from)

  Description: The functionality of this function is the same as that of SUBSTR(string,from). However, the calculation unit is byte.

Return type: bytea.

Example:

```
gaussdb=# SELECT substrb('ABCDEF',-2);
 substrb
---------
 EF
(1 row)
```

- substrb(string,from,count)

  Description: The functionality of this function is the same as that of SUBSTR(string,from,count). However, the calculation unit is byte.

  Return type: bytea.

  Example:

  ```
  gaussdb=# SELECT substrb('ABCDEF',2,2);
   substrb
  ---------
   BC
  (1 row)
  ```

- trim([leading |trailing |both] [characters] from string)

  Description: Removes the longest string consisting only of characters in **characters** (a space by default) from the start, end, or both sides of **string**.

  Return type: text.

  Example:

  ```
  gaussdb=# SELECT trim(BOTH 'x' FROM 'xTomxx');
   btrim
  -------
   Tom
  (1 row)
  gaussdb=# SELECT trim(LEADING 'x' FROM 'xTomxx');
   ltrim
  -------
   Tomxx
  (1 row)
  gaussdb=# SELECT trim(TRAILING 'x' FROM 'xTomxx');
   rtrim
  -------
   xTom
  (1 row)
  ```

- to_single_byte(char)

  Description: Converts all multi-byte characters in a string to single-byte characters.

  Return type: text.

  NOTE

  This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an A-compatible database.

- to_multi_byte(char)

  Description: Converts all single-byte characters in a string to multi-byte characters.

  Return type: text.

  NOTE

  This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an A-compatible database.

- rtrim(string [, characters])

  Description: Removes the longest string consisting only of characters in **characters** (a space by default) from the end of **string**.

  Return type: text.

  Example:

```
gaussdb=# SELECT rtrim('TRIMxxxx','x');
 rtrim
-------
 TRIM
(1 row)
```

- ltrim(string [, characters])

  Description: Removes the longest string consisting only of characters in **characters** (a space by default) from the start of **string**.

  Return type: text.

  Example:

```
gaussdb=# SELECT ltrim('xxxxTRIM','x');
 ltrim
-------
 TRIM
(1 row)
```

- upper(string)

  Description: Converts the string into the uppercase.

  Return type: text.

  Example:

```
gaussdb=# SELECT upper('tom');
 upper
-------
 TOM
(1 row)
```

- lower(string)

  Description: Converts the string into the lowercase.

  Return type: text.

  Example:

```
gaussdb=# SELECT lower('TOM');
 lower
-------
 tom
(1 row)
```

- nls_upper(string [, nlsparam])

  Description: Converts a character string to uppercase letters. You can specify a sorting rule to process special uppercase conversion rules in some languages. The format of **nlsparam** is **'nls_sort=sort_name'**, where **sort_name** is replaced by the specific sorting rule name. When the **nlsparam** parameter is not set, this function is equivalent to **upper**.

  Return type: text.

  Example:

```
gaussdb=# SELECT nls_upper('große');
 nls_upper
-----------
 GROßE
(1 row)
```



```
gaussdb=# SELECT nls_upper('große', 'nls_sort = XGerman');
 nls_upper
-----------
 GROSSE
(1 row)
```

> NOTE
>
> This function can be used only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an A-compatible database.

- nls_lower(string [, nlsparam])

  Description: Converts a character string to lowercase letters. You can specify a sorting rule to process special lowercase conversion rules in some languages. The format of **nlsparam** is **'nls_sort=sort_name'**, where **sort_name** is replaced by the specific sorting rule name. When the **nlsparam** parameter is not set, this function is equivalent to **lower**.

  Return type: text.

  Example:

```
gaussdb=# SELECT nls_lower('INDIVISIBILITY');
   nls_lower
----------------
 indivisibility
(1 row)
gaussdb=# SELECT nls_lower('INDIVISIBILITY', 'nls_sort = XTurkish');
   nls_lower
----------------
 ındıvısıbılıty
(1 row)
```

> NOTE
>
> This function can be used only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an A-compatible database.

- instr(string,substring[,position,occurrence])

  Description: Queries and returns the value of the substring position that occurs the *occurrence* (**1** by default) times from the *position* (**1** by default) in the string.

  - If the value of *position* is **0**, **0** is returned.
  - If the value of *position* is a negative number, searches backwards from the last *n*th character in the string, in which *n* indicates the absolute value of *position*.

  In this function, the calculation unit is character. One Chinese character is one character.

  Return type: integer.

  Example:

```
gaussdb=# SELECT instr('corporate floor','or', 3);
 instr
-------
     5
(1 row)
gaussdb=# SELECT instr('corporate floor','or',-3,2);
 instr
-------
     2
(1 row)
```

- initcap(string)

  Description: Capitalizes the first letter of each word in a string.

Return type: text.

Example:

```
gaussdb=# SELECT initcap('hi THOMAS');
  initcap
-----------
 Hi Thomas
(1 row)
```

NOTE

When **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s2**, if a case-insensitive character, such as Chinese, is followed by a case-sensitive character, only the first letter of the case-sensitive character is capitalized. Therefore, you are advised to set **a_format_version** to **10c** and **a_format_dev_version** to **s2**.

- ascii(string)

  Description: Indicates the ASCII code of the first character in the string.

  Return type: integer.

  Example:

  ```
  gaussdb=# SELECT ascii('xyz');
   ascii
  -------
     120
  (1 row)
  ```

- ascii2(string)

  Description: Returns the decimal code of the first character of the input string in the database character set.

  Return type: integer.

  NOTE

  This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an A-compatible database.

- asciistr(string)

  Description: Converts non-ASCII characters in the input string to *XXXX*, where *XXXX* indicates the UTF-16 code unit.

  Return type: varchar.

  NOTE

  This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an A-compatible database.

- unistr(string)

  Description: Converts the coding sequence in a string into the corresponding character. Other characters remain unchanged.

  Return type: text.

  NOTE

  - This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an A-compatible database.
  - The backslash (\) must be followed by four hexadecimal characters to indicate the coding sequence, or another backslash (\) indicates that a single backslash (\) is entered.
  - If the input parameter is of the time type, the time type is implicitly converted to the character string type.

- vsize(expr)

  Description: Returns the number of bytes of the input expression.

  Return type: int.

  > **NOTE**
  >
  > This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an A-compatible database.

- replace(string varchar, search_string varchar, replacement_string varchar)

  Description: Replaces all **search_string** in the string with **replacement_string**.

  Return type: varchar.

  Example:

  ```
  gaussdb=# SELECT replace('jack and jue','j','bl');
     replace
  ----------------
   black and blue
  (1 row)
  ```

- concat(str1,str2)

  Description: Connects str1 and str2 and returns the string. Note: **concat** calls the output function of the data type and the return value is immutable. As a result, the optimizer cannot calculate the result in advance when generating a plan. If there are performance requirements, you are advised to use the operator **||**.

  > **NOTE**
  >
  > - If **sql_compatibility** is set to **'B'** and **str1** or **str2** is set to **NULL**, the returned result is **NULL**.
  > - The return value of the concat function is of the variable-length type. When the concat function is compared with table data, the character string length is lost in the combination result. As a result, the comparison results are different.

  Return type: varchar.

  Example:

  ```
  gaussdb=# SELECT concat('Hello', ' World!');
     concat
  --------------
   Hello World!
  (1 row)
  gaussdb=# SELECT concat('Hello', NULL);
   concat
  --------
   Hello
  (1 row)
  gaussdb=# CREATE TABLE test_space(c char(10));
  CREATE TABLE
  gaussdb=# INSERT INTO test_space values('a');
  INSERT 0 1
  -- After spaces are padded, the character string is still a fixed-length character string. It is expected that the result can be found.
  gaussdb=# SELECT * FROM test_space WHERE c = 'a ';
       c
  ------------
   a
  (1 row)
  -- The combination result is a variable-length character string. The comparison fails and the result cannot be found.
  gaussdb=# SELECT * FROM test_space WHERE c = 'a' || ' ';
   c
  ```



```
---
(0 rows)
```

- chr(integer)

  Description: For the UTF-8 character set, the input is encoded as Unicode and a UTF-8 character is returned. For other character sets, an ASCII character is returned.

  Return type: text.

  NOTE

  - When **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1** in an A-compatible database, the value is truncated instead of being rounded off if the value of *integer* is a decimal.
  - For details about how to set and modify the character set and collation, see **Character Sets and Collations**.

  Example:

```
gaussdb=# SELECT chr(65);
 chr
-----
 A
(1 row)

-- In the case of UTF-8 character set
gaussdb=# SELECT chr(19968);
 chr
-----
 一
(1 row)
```

- chr(cvalue int|bigint)

  Description: Converts *cvalue* to the character of the corresponding byte order and returns the character.

  cvalue can be converted into a value of the int or bigint type. The value range is [0,2^32 – 1], corresponding to the range of unsigned int. A character array consisting of one to four bytes is returned based on the value of *n*. The byte arrays returned in different character sets are the same. However, due to different encoding rules, the result of the returned character string varies depending on the character set encoding.

  If the character set is a single-byte character set, an ASCII character is returned after *cvalue* mod 256.

  Precautions:

  – If a byte in the input *cvalue* is **0**, the output is truncated.
  – If the input does not comply with the encoding rule of the current character set, an error is reported.
  – If the input is **NULL** or **0**, **NULL** is returned.

  Return type: text.

  Example:

```
gaussdb=# SELECT chr(65);
 chr
-----
 A
(1 row)
gaussdb=# SET a_format_version='10c';
gaussdb=# SET a_format_dev_version = 's1';
gaussdb=# SELECT chr(16705);
 chr
```



```
-----
 AA
(1 row)

-- The output is truncated.
gaussdb=# SELECT chr(4259905);
 chr
-----
 A
(1 row)
```

> **NOTE**
>
> When **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1** in an A-compatible database, the chr function returns a character in the collation based on the input value. If the current character set for database encoding is a multi-byte character set, the return value contains one to four bytes. If the current character set for database encoding is a single-byte character set, the return value is a single byte obtained by performing the mod 256 operation on the input value. Otherwise, if the current character set for database encoding is UTF-8, the input is encoded as Unicode and a UTF-8 character is returned. For other character sets, an ASCII character is returned.

- nchr(cvalue int|bigint)

  Description: Returns the characters corresponding to the input parameter in the national character set. The GUC parameter **nls_nchar_characterset** specifies the country character set. Only AL16UTF16 and UTF8 are supported. This function is valid only when the value of the GUC parameter **a_format_version** is **10c** and that of **a_format_dev_version** is **s4** in an A-compatible database. If **nls_nchar_characterset** is set to **AL16UTF16** and the input parameter exceeds two bytes, the input parameter will be truncated and the lowest two bytes will be retained. If **nls_nchar_characterset** is set to **UTF8** and the input parameter exceeds three bytes, the value **0** is used.

  Parameter: cvalue. **cvalue** can be converted to the int or bigint type. The value range is [0,2^32 – 1], corresponding to the range of unsigned int. If the input parameter contains the decimal part, the decimal part will be removed.

  Return type: NVARCHAR2.

CAUTION

- The byte length of the return value of the function is different from that in database A.
- The return value of the function is restricted by the database character set. In different database character sets, if there is no mapping table for converting UTF8 to the database character set or the mapping table does not contain the UTF8 code, the current database character set does not support the UTF8 character corresponding to the input parameter. As a result, the result returned by the nchr(cvalue int|bigint) function is inconsistent with that in database A.
- If the current database character set does not support the UTF8 character corresponding to the input parameter, or the national character set is UTF8 but the input parameter does not comply with the UTF8 format, the byte array corresponding to the input parameter is returned. If a single byte is in the range of [0x00,0x7F], an ASCII character is returned. If a single byte is in the range of [0x80,0xFF], a question mark (?) is returned.
- For details about how to set and modify the character set and collation, see **Character Sets and Collations**.

- regexp_substr(source_char, pattern)

  Description: Extracts substrings from a regular expression. If the SQL syntax is compatible with products A and B and the value of the GUC parameter **behavior_compat_options** contains **aformat_regexp_match**, the period (.) cannot match the **'\n'** character. If **aformat_regexp_match** is not contained, the period (.) matches the **'\n'** character by default.

  Return type: text.

  Example:

```
gaussdb=# SELECT regexp_substr('500 Hello World, Redwood Shores, CA', ',[^,]+,')
"REGEXPR_SUBSTR";
  REGEXPR_SUBSTR
------------------
 , Redwood Shores,
(1 row)
```

- regexp_replace(string, pattern, replacement [,flags])

  Description: Replaces substrings matching the POSIX regular expression. The source string is returned unchanged if there is no match to the pattern. If there is a match, the source string is returned with the replacement string substituted for the matching substring.

  The replacement string can contain **\n**, where **n** is 1 through 9, to indicate that the source substring matching the *n*th parenthesized sub-expression of the pattern should be inserted, and it can contain **\&** to indicate that the substring matching the entire pattern should be inserted.

  The optional **flags** parameter contains zero or multiple single-letter flags that change the behavior of a function. The options supported by **flags** and description are described in **Table 7-31**.

  Return type: varchar.

  Example:

```
gaussdb=# SELECT regexp_replace('Thomas', '.[mN]a.', 'M');
 regexp_replace
```



```
----------------
 ThM
(1 row)
gaussdb=# SELECT regexp_replace('foobarbaz','b(..)', E'X\\1Y', 'g') AS
RESULT;
   result
-------------
 fooXarYXazY
(1 row)
```

- repexp_replace(string text, pattern text [, replacement text [, position int [, occurrence int [, flags text]]]])

  Description: Replaces substrings matching the POSIX regular expression. The source string is returned unchanged if there is no match to the pattern. If there is a match, the source string is returned with the replacement string substituted for the matching substring.

  Parameter description:

  - **string**: source character string used for matching.
  - **pattern**: regular expression pattern string used for matching.
  - **replacement**: character string used to replace the matched substring. This parameter is optional. If no parameter value is specified or the parameter value is null, an empty string is used for replacement.
  - **position**: start character of the source string used for matching. This parameter is optional. The default value is **1**.
  - **occurrence**: sequence number of the matched substring to be replaced. This parameter is optional. The default value is **1**, indicating that the first matched substring is replaced.
  - **flags**: contains zero or multiple single-letter flags that change the matching behavior of the function. This parameter is optional. The options supported by **flags** and description are described in **Table 7-31**.

  NOTE

  When **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1** in an A-compatible database, the default value of **occurrence** is **0**, indicating that all matched substrings are replaced. In addition, the **pattern** parameter that ends with a backslash (\) is valid.

  Return type: text.

  Example:

```
gaussdb=# SELECT regexp_replace('foobarbaz','b(..)', E'X\\1Y', 2, 2, 'n') AS RESULT;
result
------------
foobarXazY
(1 row)
```

- concat_ws(sep text, str"any" [, str"any" [, ...]])

  Description: Uses the first parameter as the separator, which is associated with all following parameters. The **NULL** parameter is ignored.

NOTICE

- If the first parameter value is **NULL**, the returned result is **NULL**.
- If the first parameter is provided but the parameter value is an empty string ('') and the SQL compatibility mode of the database is set to **A**, the returned result is **NULL**. This is because the A-compatible mode treats the empty string ('') as **NULL**. To resolve this problem, you can change the SQL compatibility mode of the database to B, C, or PG.

Return type: text.

Example:

```
gaussdb=# SELECT concat_ws(',', 'ABCDE', 2, NULL, 22);
 concat_ws
------------
 ABCDE,2,22
(1 row)
```

- nlssort(string text, sort_method text)

  Description: Returns the encoding value of a string in the sorting mode specified by **sort_method**. The encoding value can be used for sorting and determines the sequence of the string in the sorting mode. Currently, **sort_method** can be set to **nls_sort=schinese_pinyin_m** or **nls_sort=generic_m_ci**. **nls_sort=generic_m_ci** supports only the case-insensitive order for English characters.

  String type: text.

  sort_method type: text

  Return type: text.

  Example:

```
gaussdb=# CREATE TABLE test(a text);

gaussdb=# INSERT into test(a) VALUES ('abC');

gaussdb=# INSERT into test(a) VALUES ('abC');

gaussdb=# INSERT into test(a) VALUES ('abc');

gaussdb=# SELECT * FROM test ORDER BY nlssort(a,'nls_sort=schinese_pinyin_m');
   a
--------
 abc
 abC
 abC
(3 rows)

gaussdb=# SELECT * FROM test ORDER BY nlssort(a, 'nls_sort=generic_m_ci');
   a
--------
 abC
 abc
 abC
(3 rows)

gaussdb=# DROP TABLE test;
```

- convert(string bytea, src_encoding name, dest_encoding name)

  Description: Converts the string to **dest_encoding**. **src_encoding** specifies the source code encoding. The string must be valid in this encoding.

  Return type: bytea.

Example:

```
gaussdb=# SELECT convert('text_in_utf8', 'UTF8', 'GBK');
         convert
----------------------------
 \x746578745f696e5f75746638
(1 row)
```

NOTE

If the rule for converting between source to target encoding (for example, GBK and LATIN1) does not exist, the string is returned without conversion. See the pg_conversion system catalog for details. **server_encoding** is specified during database initialization.

Example:

```
gaussdb=# SHOW server_encoding;
 server_encoding
-----------------
 LATIN1
(1 row)

gaussdb=# SELECT convert_from('some text', 'GBK');
 convert_from
--------------
 some text
(1 row)

db_latin1=# SELECT convert_to('some text', 'GBK');
      convert_to
----------------------
 \x736f6d652074657874
(1 row)

db_latin1=# SELECT convert('some text', 'GBK', 'LATIN1');
       convert
----------------------
 \x736f6d652074657874
(1 row)
```

- convert(expr, USING transcoding_name)

  Description: Converts a parameter, which can be a character or a number, into a character string of the transcoding_name type, and returns the character string.

  Return type: text.

  Example:

  ```
  gaussdb=# SELECT convert('asdas' using 'gbk');
   convert
  ---------
   asdas
  (1 row)
  ```

  NOTE

  This function takes effect only in databases in B compatibility mode.

- convert_from(string bytea, src_encoding name)

  Description: Converts a string using the coding mode of the database.

  src_encoding specifies the source code encoding. The string must be valid in this encoding.

  Return type: text.

  Example:

  ```
  gaussdb=# SELECT convert_from('text_in_utf8', 'UTF8');
   convert_from
  ```



```
--------------
 text_in_utf8
(1 row)
```

- convert_to(string text, dest_encoding name)

  Description: Converts a string to **dest_encoding**.

  Return type: bytea.

  Example:

  ```
  gaussdb=# SELECT convert_to('some text', 'UTF8');
       convert_to
  ----------------------
   \x736f6d652074657874
  (1 row)
  ```

- string [NOT] LIKE pattern [ESCAPE escape-character]

  Description: Specifies the pattern matching function.

  If the pattern does not include a percentage sign (%) or an underscore (_), this mode represents itself only. In this case, the behavior of LIKE is the same as the equal operator. The underscore (_) in the pattern matches any single character while one percentage sign (%) matches no or multiple characters.

  To match with underscores (_) or percent signs (%), corresponding characters in **pattern** must lead escape characters. The default escape character is a backward slash (\) and can be specified using the **ESCAPE** clause. To match with escape characters, enter two escape characters.

  Return type: Boolean

  Example:

  ```
  gaussdb=# SELECT 'AA_BBCC' LIKE '%A@_B%' ESCAPE '@' AS RESULT;
   result
  --------
   t
  (1 row)
  gaussdb=# SELECT 'AA_BBCC' LIKE '%A@_B%' AS RESULT;
   result
  --------
   f
  (1 row)
  gaussdb=# SELECT 'AA@_BBCC' LIKE '%A@_B%' AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- REGEXP_LIKE(source_string, pattern [, match_parameter])

  Description: Indicates the mode matching function of a regular expression.

  source_string indicates the source string and **pattern** indicates the matching pattern of the regular expression. **match_parameter** indicates the matching items and the values are as follows:

  – 'i': case-insensitive

  – 'c': case-sensitive

  – 'n': allowing the metacharacter "." in a regular expression to be matched with a linefeed.

  – 'm': allows **source_string** to be regarded as multiple rows.

  If **match_parameter** is ignored, **case-sensitive** is enabled by default, "." is not matched with a linefeed, and **source_string** is regarded as a single row.

  Return type: Boolean

Example:

```
gaussdb=# SELECT regexp_like('ABC', '[A-Z]');
 regexp_like
-------------
 t
(1 row)
gaussdb=# SELECT regexp_like('ABC', '[D-Z]');
 regexp_like
-------------
 f
(1 row)
gaussdb=# SELECT regexp_like('ABC', '[a-z]','i');
 regexp_like
-------------
 t
(1 row)
```

- format(formatstr text [, str"any" [, ...]])

  Description: Formats a string.

  Return type: text.

  Example:

  ```
  gaussdb=# SELECT format('Hello %s, %1$s', 'World');
        format
  --------------------
   Hello World, World
  (1 row)
  ```

- md5(string)

  Description: Encrypts a string in MD5 mode and returns a value in hexadecimal form.

  NOTE

  The MD5 encryption algorithm is not recommended because it has lower security and poses security risks.

  Return type: text.

  Example:

  ```
  gaussdb=# SELECT md5('ABC');
               md5
  ----------------------------------
   902fbdd2b1df0c4f70b4a5d23525e932
  (1 row)
  ```

- sha(string) / sha1(string)

  Description: Encrypts a string using SHA1 and returns a hexadecimal number. The sha and sha1 functions are the same.

  NOTE

  - The SHA1 encryption algorithm is not recommended because it has lower security and poses security risks.
  - This function is valid only when GaussDB is compatible with the MY type (that is, **sql_compatibility** is set to **'B'**).

  Return type: text.

  Example:

  ```
  gaussdb=# SELECT sha('ABC');
                   sha
  ------------------------------------------
   3c01bdbb26f358bab27f267924aa2c9a03fcfdb8
  (1 row)
  ```



```
gaussdb=# SELECT sha1('ABC');
                 sha1
------------------------------------------
 3c01bdbb26f358bab27f267924aa2c9a03fcfdb8
(1 row)
```

- sha2(string, hash_length)

  Description: Encrypts a string in SHA2 mode and returns a value in hexadecimal form.

  hash_length: corresponds to an SHA2 algorithm. The value can be **0** (SHA-256), **224** (SHA-224), **256** (SHA-256), **384** (SHA-384), or **512** (SHA-512). For other values, **NULL** is returned.

  NOTE

  - The SHA-224 encryption algorithm is not recommended because it has lower security and poses security risks.
  - The SHA2 function records hash plaintext in logs. Therefore, you are advised not to use this function to encrypt sensitive information such as keys.
  - This function is valid only when GaussDB is compatible with the MY type (that is, **sql_compatibility** is set to **'B'**).

  Return type: text.

  Example:

```
gaussdb=# SELECT sha2('ABC',224);
                          sha2
------------------------------------------------------------
 107c5072b799c4771f328304cfe1ebb375eb6ea7f35a3aa753836fad
(1 row)
gaussdb=# SELECT sha2('ABC',256);
                            sha2
----------------------------------------------------------------
 b5d4045c3f466fa91fe2cc6abe79232a1a57cdf104f7a26e716e0a1e2789df78
(1 row)
gaussdb=# SELECT sha2('ABC',0);
                            sha2
----------------------------------------------------------------
 b5d4045c3f466fa91fe2cc6abe79232a1a57cdf104f7a26e716e0a1e2789df78
(1 row)
```

- decode(string text, format text)

  Description: Decodes binary data from textual representation.

  Return type: bytea.

  Example:

```
gaussdb=# SELECT decode('MTIzAAE=', 'base64');
   decode
--------------
 \x3132330001
(1 row)
```

- similar_escape(pat text, esc text)

  Description: Converts a regular expression of the SQL:2008 style to the POSIX style.

  Return type: text.

  Example:

```
gaussdb=# SELECT similar_escape('\s+ab','2');
 similar_escape
----------------
 ^(?:\\s+ab)$
(1 row)
```


- find_in_set(text, set)

  Description: Finds the position of a given member in a set, counting from 1. If no record is found, 0 is returned.

  Return type: int2

  Example:

```
gaussdb=# CREATE DATABASE gaussdb_b WITH DBCOMPATIBILITY 'B';
gaussdb=# \c gaussdb_b
gaussdb_b=# CREATE TABLE employee (
  name text,
  site SET('beijing','shanghai','nanjing','wuhan')
);

gaussdb_b=# INSERT INTO employee values('zhangsan', 'beijing,nanjing');
gaussdb_b=# INSERT INTO employee values('zhangsan2', 'beijing,wuhan');

gaussdb_b=# SELECT site, find_in_set('wuhan', site) from employee;
      site       | find_in_set
-----------------+-------------
 beijing,nanjing |           0
 beijing,wuhan   |           2
(2 rows)

gaussdb_b=# DROP TABLE employee;
gaussdb_b=# \c postgres
gaussdb=# DROP DATABASE gaussdb_b;
```

- find_in_set(str, strlist)

  Description: Queries whether the **strlist** field contains **str**. If yes, the position of **str** in the **strlist** field is returned. The inputs are **str** and **strlist**. **str** indicates the character string to be queried. **strlist** is a set of character strings separated by commas (,). If **str** is not in **strlist** or **strlist** is an empty string, the return value is **0**.

  The parameters are described as follows.

  Table 7-33 Parameters

Parameter	Type	Description
str	text	Target string
strlist	text	A set of character strings

  Return type: int.

  Example:

```
gaussdb=# SELECT find_in_set('ee','a,ee,c');
 find_in_set
-------------
           2
(1 row)
```

- encode(data bytea, format text)

  Description: Encodes binary data into a textual representation.

  Return type: text.

  Example:


```
gaussdb=# SELECT encode(E'123\\000\\001', 'base64');
  encode
----------
 MTIzAAE=
(1 row)
```

- strcmp(expr1,expr2)

  Description: Compares two input strings based on the current character order. If the strings are the same, **0** is returned. If the first string is smaller than the second string, **-1** is returned. Otherwise, **1** is returned.

  The parameters are described as follows.

Parameter	Type	Description
expr1/expr2	Character types: CHAR, VARCHAR, NVARCHAR2, and TEXT. Binary type: BYTEA. Numeral types: TINYING [UNSIGNED], SMALLINT [UNSIGNED], INTEGER [UNSIGNED], BIGINT [UNSIGNED], FLOAT4, FLOAT8, and NUMERIC. Date and time types: DATE, TIME WITHOUT TIME ZONE, DATETIME, and TIMESTAMPTZ.	Character string involved in comparison.

  Return type: integer.

  Example:

```
-- Switch to the B-compatible database.
gaussdb=# CREATE DATABASE gaussdb_m dbcompatibility='B';
gaussdb=# \c gaussdb_m

-- Set the compatible version control parameter to enable the function of specifying the collation for
constant character strings.
gaussdb_m=# SET b_format_version='5.7';
gaussdb_m=# SET b_format_dev_version='s2';

gaussdb_m=# SELECT strcmp('abc', 'ABC');
 strcmp
--------
      0
(1 row)

gaussdb_m=# SELECT strcmp('abc     ', 'abc');
 strcmp
--------
      0
(1 row)

gaussdb_m=# SELECT strcmp('1', 1);
 strcmp
--------
```



```
     0
(1 row)

gaussdb_m=# SELECT strcmp(123, 2);
 strcmp
--------
    -1
(1 row)
```

NOTE

- The strcmp function takes effect only when **sql_compatibility** is set to **'B'**.
- Since the version whose **b_format_version** is set to **'5.7'** and **b_format_dev_version** is set to **'s1'**, the SQL_MODE parameter **pad_char_to_full_length** specifies whether to add spaces at the end of the char type, which affects the strcmp comparison result. For details, see **Table 7-8**.
- Since the version whose **b_format_version** is set to **'5.7'** and **b_format_dev_version** is set to **'s1'**, the behavior of the character, binary, numeric, and date and time types is M-compatible, which affects the strcmp comparison result. For details, see **Data Types**. For the floating-point type in the numeric type, the precision may be different from that in MySQL due to different connection parameter settings. Therefore, this scenario is not recommended, or the numeric type is used instead. For details, see **Connection Parameters**.
- Since the version whose **b_format_version** is set to **'5.7'** and **b_format_dev_version** is set to **'s2'**, constant character string obtaining collation is supported. The collation affects the strcmp comparison result. For details, see the SET NAMES syntax in **SET**. For details about the rules for combining different collations of the character type, see **Rules for Combining Character Sets and Collations**.

NOTE

- For a string containing linefeed characters, for example, a string consisting of a linefeed character and a space, the value of **LENGTH** and **LENGTHB** in GaussDB is 2.
- In GaussDB, *n* in the CHAR(n) type indicates the number of characters. Therefore, for multiple-octet coded character sets, the length returned by the LENGTHB function may be longer than *n*.
- GaussDB supports multiple types of databases, including A-, B-, C-, and PG-compatible. If the database type is not specified, A-compatible mode is used by default for GaussDB. The lexical analyzer of A-compatible database is different from that of the other three databases. In A-compatible database, an empty character string is considered as **NULL**. Therefore, when a type A database is used, if an empty string is used as a parameter in the preceding character operation function, no output is displayed. For example:

  ```
  gaussdb=# SELECT translate('12345','123','');
  translate
  -----------

  (1 row)
  ```

  This is because the kernel checks whether the input parameter contains **NULL** before calling the corresponding function. If the input parameter contains **NULL**, the kernel does not call the corresponding function. As a result, no output is displayed. In PG mode, the processing of character strings is the same as that of PostgreSQL. Therefore, the preceding problem does not occur.

Extension Functions and Operators

- pkg_bpchar_opc()

  Description: Serves as an extension API to add the comparison operator between bpchar and text or between text and bpchar policies, so as to solve the problem that indexes cannot be matched when data of the bpchar and text types is compared. Only the system administrator can install extensions.

Example:

Compare the bpchar type with the text type (initial state, forward compatibility).

```
/*
Create a table and initialize data.
*/
gaussdb=# CREATE TABLE logs_nchar(log_id nchar(16), log_message text);
gaussdb=# INSERT INTO logs_nchar SELECT GENERATE_SERIES(1,100000),MD5(RANDOM());
gaussdb=# INSERT INTO logs_nchar VALUES ('FE306991300002  ','002');
gaussdb=# CREATE INDEX idx_nchar_logid on logs_nchar(log_id);
gaussdb=# VACUUM ANALYZE logs_nchar;

/*
If no extension is installed, when nchar and text are compared, nchar is implicitly converted to text because there is no bpchar or text index operator. That is, the fixed-length character type is converted to the variable-length character type. As a result, the execution plan changes and the index cannot be matched.
*/
gaussdb=# EXPLAIN SELECT * FROM logs_nchar WHERE log_id = RPAD(TRIM('FE306991300002  '),16,'
');
                            QUERY PLAN
------------------------------------------------------------------
 Seq Scan on logs_nchar  (cost=0.00..2236.01 rows=500 width=50)
   Filter: ((log_id)::text = 'FE306991300002  '::text)
(2 rows)

/*
The log_id column in the logs_nchar table is of nchar(16) type. The inserted data is 'FE306991300002  ', which is implicitly converted to text. During comparison, spaces are deleted, that is, 'FE306991300002'='FE306991300002  '. Therefore, the data is not matched.
*/
gaussdb=# SELECT * FROM logs_nchar WHERE log_id = RPAD(TRIM('FE306991300002  '),16,' ');
 log_id | log_message
--------+-------------
(0 rows)

/*
Delete the table.
*/
gaussdb=# DROP TABLE logs_nchar;
```

Compare the bpchar type with the text type (the pkg_bpchar_opc extension is installed, which is A-compatible).

```
/*
Create a table and initialize data.
*/
gaussdb=# CREATE TABLE logs_nchar(log_id nchar(16), log_message text);
gaussdb=# INSERT INTO logs_nchar SELECT GENERATE_SERIES(1,100000),MD5(RANDOM());
gaussdb=# INSERT INTO logs_nchar VALUES ('FE306991300002  ','002');
gaussdb=# CREATE INDEX idx_nchar_logid on logs_nchar(log_id);
gaussdb=# VACUUM ANALYZE logs_nchar;

/*
The system administrator installs the pkg_bpchar_opc extension. The comparison operators of the bpchar and text types and index-related content are added to the database.
*/
gaussdb=# CREATE EXTENSION pkg_bpchar_opc;
CREATE EXTENSION

/*
In this case, when log_id is implicitly converted to the bpchar type and compared with the text type, the comparison operator and index information can be found, and the index can be matched.
*/
gaussdb=# EXPLAIN SELECT * FROM logs_nchar WHERE log_id = RPAD(TRIM('FE306991300002  '),16,'
');
                                QUERY PLAN
---------------------------------------------------------------------------------
 [Bypass]
```



```
 Index Scan using idx_nchar_logid on logs_nchar  (cost=0.00..8.27 rows=1 width=50)
   Index Cond: (log_id = 'FE306991300002  '::text)
(3 rows)

gaussdb=# SELECT * FROM logs_nchar WHERE log_id = RPAD(TRIM('FE306991300002  '),16,' ');
      log_id      | log_message
------------------+-------------
 FE306991300002   | 002
(1 row)

/*
Delete the table and extension.
*/
gaussdb=# DROP TABLE logs_nchar;
gaussdb=# DROP EXTENSION pkg_bpchar_opc;
```

Compare the text type with the bpchar type (initial state, forward compatibility).

```
/*
Create a table and initialize data.
*/
gaussdb=# CREATE TABLE logs_text(log_id nchar(16), log_message text);
gaussdb=# INSERT INTO logs_text SELECT GENERATE_SERIES(1,100000),MD5(RANDOM());
gaussdb=# INSERT INTO logs_text VALUES ('FE306991300002  ','002');
gaussdb=# CREATE INDEX idx_text_logid on logs_text(log_id);
gaussdb=# VACUUM ANALYZE logs_text;

gaussdb=# SELECT * FROM logs_text WHERE log_id = 'FE306991300002  '::bpchar;
 log_id | log_message
--------+-------------
(0 rows)

gaussdb=# EXPLAIN SELECT * FROM logs_text WHERE log_id = 'FE306991300002  '::bpchar;
                                QUERY PLAN
-------------------------------------------------------------------------------
 [Bypass]
 Index Scan using idx_text_logid on logs_text  (cost=0.00..8.27 rows=1 width=37)
   Index Cond: (log_id = 'FE306991300002'::text)
(3 rows)

/*
Delete the table.
*/
gaussdb=# DROP TABLE logs_text;
```

Compare the text type with the bpchar type (the pkg_bpchar_opc extension is installed, which is A-compatible).

```
/*
Create a table and initialize data.
*/
gaussdb=# CREATE TABLE logs_text(log_id nchar(16), log_message text);
gaussdb=# INSERT INTO logs_text SELECT GENERATE_SERIES(1,100000),MD5(RANDOM());
gaussdb=# INSERT INTO logs_text VALUES ('FE306991300002  ','002');
gaussdb=# CREATE INDEX idx_text_logid on logs_text(log_id);
gaussdb=# VACUUM ANALYZE logs_text;

gaussdb=# CREATE EXTENSION pkg_bpchar_opc;
CREATE EXTENSION

gaussdb=# SELECT * FROM logs_text WHERE log_id = 'FE306991300002  '::bpchar;
      log_id      | log_message
------------------+-------------
 FE306991300002   | 002
(1 row)

gaussdb=# EXPLAIN SELECT * FROM logs_text WHERE log_id = 'FE306991300002  '::bpchar;
                                QUERY PLAN
-------------------------------------------------------------------------------
 [Bypass]
 Index Scan using idx_text_logid on logs_text  (cost=0.00..8.27 rows=1 width=38)
```



```
  Index Cond: (log_id = 'FE306991300002  '::bpchar)
(3 rows)

/*
Delete the table and extension.
*/
gaussdb=# DROP TABLE logs_text;
gaussdb=# DROP EXTENSION pkg_bpchar_opc;
```

Compare the hash join and the text type with the bpchar type (initial state, forward compatibility).

```
/*
Create a table and initialize data.
*/
gaussdb=# CREATE TABLE logs_varchar2(log_id varchar2, log_message text);
gaussdb=# INSERT INTO logs_varchar2 VALUES ('FE306991300002  ','002');
gaussdb=# INSERT INTO logs_varchar2 VALUES ('FE306991300003  ','003');
gaussdb=# INSERT INTO logs_varchar2 VALUES ('FE306991300004  ','004');
gaussdb=# VACUUM ANALYZE logs_varchar2;

gaussdb=# CREATE TABLE logs_char(log_id char(16), log_message text);
gaussdb=# INSERT INTO logs_char VALUES ('FE306991300002  ','002');
gaussdb=# INSERT INTO logs_char VALUES ('FE306991300003  ','003');
gaussdb=# INSERT INTO logs_char VALUES ('FE306991300004  ','004');
gaussdb=# VACUUM ANALYZE logs_char;

gaussdb=# SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id = t2.log_id;
     log_id      | log_message |      log_id      | log_message
-----------------+-------------+------------------+-------------
 FE306991300002  | 002         | FE306991300002   | 002
 FE306991300003  | 003         | FE306991300003   | 003
 FE306991300004  | 004         | FE306991300004   | 004
(3 rows)

gaussdb=# EXPLAIN SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id = t2.log_id;
                           QUERY PLAN
---------------------------------------------------------------------------
 Hash Join  (cost=1.07..2.14 rows=3 width=42)
   Hash Cond: ((t1.log_id)::bpchar = t2.log_id)
   ->  Seq Scan on logs_varchar2 t1  (cost=0.00..1.03 rows=3 width=21)
   ->  Hash  (cost=1.03..1.03 rows=3 width=21)
         ->  Seq Scan on logs_char t2  (cost=0.00..1.03 rows=3 width=21)
(5 rows)

gaussdb=# SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id = 'FE306991300002  ';
     log_id      | log_message |      log_id      | log_message
-----------------+-------------+------------------+-------------
 FE306991300002  | 002         | FE306991300002   | 002
 FE306991300002  | 002         | FE306991300003   | 003
 FE306991300002  | 002         | FE306991300004   | 004
(3 rows)

/*
Delete the table.
*/
gaussdb=# DROP TABLE logs_varchar2;
gaussdb=# DROP TABLE logs_char;
```

Compare the hash join and the text type with the bpchar type (the pkg_bpchar_opc extension is installed, which is A-compatible).

```
/*
Create a table and initialize data.
*/
gaussdb=# CREATE TABLE logs_varchar2(log_id varchar2, log_message text);
gaussdb=# INSERT INTO logs_varchar2 VALUES ('FE306991300002  ','002');
gaussdb=# INSERT INTO logs_varchar2 VALUES ('FE306991300003  ','003');
gaussdb=# INSERT INTO logs_varchar2 VALUES ('FE306991300004  ','004');
gaussdb=# VACUUM ANALYZE logs_varchar2;

gaussdb=# CREATE TABLE logs_char(log_id char(16), log_message text);
```



```
gaussdb=# INSERT INTO logs_char VALUES ('FE306991300002  ','002');
gaussdb=# INSERT INTO logs_char VALUES ('FE306991300003  ','003');
gaussdb=# INSERT INTO logs_char VALUES ('FE306991300004  ','004');
gaussdb=# VACUUM ANALYZE logs_char;

gaussdb=# CREATE EXTENSION pkg_bpchar_opc;
CREATE EXTENSION

/*
This format is not recommended. After the extension is installed, the varchar2 type of log_id in the t1 table is implicitly converted to the text type. When it is compared with the log_id in the t2 table, the char type of log_id in the t2 table is implicitly converted to the bpchar type. In this case, spaces after log_id is removed by the database, that is, 'FE306991300002'='FE306991300002  '. Therefore, no data is matched.
*/
/*
Incorrect example:
*/
gaussdb=# SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id = t2.log_id;
 log_id | log_message | log_id | log_message
--------+-------------+--------+-------------
(0 rows)
gaussdb=# explain SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id = t2.log_id;
                            QUERY PLAN
------------------------------------------------------------------------------
 Hash Join  (cost=1.07..2.14 rows=3 width=42)
   Hash Cond: ((t1.log_id)::text = t2.log_id)
   ->  Seq Scan on logs_varchar2 t1  (cost=0.00..1.03 rows=3 width=21)
   ->  Hash  (cost=1.03..1.03 rows=3 width=21)
         ->  Seq Scan on logs_char t2  (cost=0.00..1.03 rows=3 width=21)
(5 rows)

gaussdb=# SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id = 'FE306991300002  ';
      log_id      | log_message |      log_id      | log_message
------------------+-------------+------------------+-------------
 FE306991300002   | 002         | FE306991300002   | 002
 FE306991300002   | 002         | FE306991300003   | 003
 FE306991300002   | 002         | FE306991300004   | 004
(3 rows)

/*
This format is recommended to avoid the following problems: the data type of log_id in the t1 table is converted to the text type, spaces are reserved during comparison, and data cannot be matched when the data type of log_id in the t2 table is compared with the data type of log_id in the t1 table. The type of t1 table is forcibly converted to the bpchar type before the extension is installed, that is, 'FE306991300002' = 'FE306991300002'. Therefore, data is matched.
*/
/*
Correct example:
/
gaussdb=# SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id::bpchar = t2.log_id;
      log_id      | log_message |      log_id      | log_message
------------------+-------------+------------------+-------------
 FE306991300002   | 002         | FE306991300002   | 002
 FE306991300003   | 003         | FE306991300003   | 003
 FE306991300004   | 004         | FE306991300004   | 004
(3 rows)

/*
The execution plan is the same as that before the extension is installed.
*/
gaussdb=# EXPLAIN SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id::bpchar =
t2.log_id;
                            QUERY PLAN
------------------------------------------------------------------------------
 Hash Join  (cost=1.07..2.14 rows=3 width=42)
   Hash Cond: ((t1.log_id)::bpchar = t2.log_id)
   ->  Seq Scan on logs_varchar2 t1  (cost=0.00..1.03 rows=3 width=21)
   ->  Hash  (cost=1.03..1.03 rows=3 width=21)
```



```
       ->  Seq Scan on logs_char t2  (cost=0.00..1.03 rows=3 width=21)
(5 rows)

gaussdb=# SELECT * FROM logs_varchar2 t1, logs_char t2 WHERE t1.log_id = 'FE306991300002  ';
     log_id      | log_message |      log_id      | log_message
-----------------+-------------+------------------+-------------
 FE306991300002  | 002         | FE306991300002   | 002
 FE306991300002  | 002         | FE306991300003   | 003
 FE306991300002  | 002         | FE306991300004   | 004
(3 rows)

/*
Delete the table and extension.
*/
gaussdb=# DROP TABLE logs_varchar2;
gaussdb=# DROP TABLE logs_char;
gaussdb=# DROP EXTENSION pkg_bpchar_opc;
```

NOTE

- This solves the problem that data and indexes cannot be properly matched when equality matching is performed between the bpchar type (containing multiple spaces) and the text type.
- The UB-tree and B-tree are involved. The comparison symbols include >, >=, <, <=, and <>.
- The impact scope involves implicit conversion between character types. For example, when a variable-length data type is compared with a fixed-length data type, the variable-length data type is preferentially converted to the text type instead of the original bpchar type.
- The pkg_bpchar_opc extension is disabled by default. You can check whether the extension is enabled in the pg_extension system catalog. If the extension data exists, the extension is enabled. If the extension data does not exist, the extension is disabled. When extension is disabled, forward compatibility is maintained. When extension is enabled, compatibility with database A is maintained. The extended function is for internal use only. You are advised not to use it.
- In the example, the table structure uses **log_id** as the index and has two columns: **log_id** and **log_message**. The table name is followed by the **log_id** column type. For example, if the table name is **logs_text**, the **log_id** column type is text.

Table 7-34 Functions supported by pkg_bpchar_opc

API	Description
pg_catalog.bpchar_text_lt	Compares the bpchar type with the text type to check whether the value on the left is less than the value on the right.
pg_catalog.bpchar_text_le	Compares the bpchar type with the text type to check whether the value on the left is less than or equal to the value on the right.
pg_catalog.bpchar_text_eq	Compares the bpchar type with the text type to check whether the value on the left is equal to the value on the right.
pg_catalog.bpchar_text_ge	Compares the bpchar type with the text type to check whether the value on the left is greater than or equal to the value on the right.

API	Description
pg_catalog.bpchar_text_gt	Compares the bpchar type with the text type to check whether the value on the left is greater than the value on the right.
pg_catalog.bpchar_text_ne	Compares the bpchar type with the text type to check whether the value on the left is different from the value on the right.
pg_catalog.bpchar_text_cmp	Specifies that the index of the bpchar and text types supports comparison functions.
pg_catalog.text_bpchar_lt	Compares the text type with the bpchar type to check whether the value on the left is less than the value on the right.
pg_catalog.text_bpchar_le	Compares the text type with the bpchar type to check whether the value on the left is less than or equal to the value on the right.
pg_catalog.text_bpchar_eq	Compares the text type with the bpchar type to check whether the value on the left is equal to the value on the right.
pg_catalog.text_bpchar_ge	Compares the text type with the bpchar type to check whether the value on the left is greater than or equal to the value on the right.
pg_catalog.text_bpchar_gt	Compares the text type with the bpchar type to check whether the value on the left is greater than the value on the right.
pg_catalog.text_bpchar_ne	Compares the text type with the bpchar type to check whether the value on the left is different from the value on the right.
pg_catalog.text_bpchar_cmp	Specifies that the index of the text and bpchar types supports comparison functions.
pg_catalog.hashbpchartext	Specifies that the hash of the bpchar and text types supports comparison functions.
pg_catalog.hashtextbpchar	Specifies that the hash of the text and bpchar types supports comparison functions.

- bpcharlikebpchar(BPCHAR, BPCHAR)

  Description: Determines whether the BPCHAR character string of the first input parameter is LIKE and whether the BPCHAR character string of the second input parameter is LIKE. The LIKE operator of the BPCHAR and BPCHAR types is added to solve the problem that the correct result set cannot be returned when data of the BPCHAR and BPCHAR types is compared. To enable the ~~ operator, ensure that the value of **behavior_compat_options** contains the **enable_bpcharlikebpchar_compare** configuration item.

  Return value type: Boolean

Value range:

- **t**: Two parameters of the BPCHAR type are matched.
- **f**: Two parameters of the BPCHAR type are not matched.

Example:

```
gaussdb=#  SELECT bpcharlikebpchar('455'::BPCHAR(10), '455  '::BPCHAR);
 bpcharlikebpchar
------------------
 f
(1 row)
gaussdb=#  SELECT bpcharlikebpchar('455 '::BPCHAR(10), '455 '::BPCHAR(10));
 bpcharlikebpchar
------------------
 t
(1 row)
gaussdb=#  SELECT bpcharlikebpchar('455 '::BPCHAR(10), '455'::BPCHAR(10));
 bpcharlikebpchar
------------------
 t
(1 row)
gaussdb=#  SELECT bpcharlikebpchar('455 '::BPCHAR(10), '455'::BPCHAR(11));
 bpcharlikebpchar
------------------
 f
(1 row)
gaussdb=#  CREATE TABLE op_test (
   col BPCHAR(2) DEFAULT NULL
 );
CREATE TABLE
gaussdb=#  CREATE INDEX op_index ON op_test(col);
CREATE INDEX
gaussdb=#  INSERT INTO op_test VALUES ('a');
INSERT 0 1
gaussdb=#  INSERT INTO op_test VALUES ('1');
INSERT 0 1
gaussdb=#  INSERT INTO op_test VALUES ('11');
INSERT 0 1
gaussdb=#  INSERT INTO op_test VALUES ('12');
INSERT 0 1
gaussdb=#  INSERT INTO op_test VALUES ('sd');
INSERT 0 1
gaussdb=#  INSERT INTO op_test VALUES ('aa');
INSERT 0 1
gaussdb=#  SHOW behavior_compat_options;
 behavior_compat_options
-------------------------
(1 row)
-- If behavior_compat_options does not contain enable_bpcharlikebpchar_compare, the latest
bpcharlikebpchar operator is not enabled and the result set returned by the matching between
bpchars is not the same as expected (all data should be returned in normal cases).
gaussdb=#  EXPLAIN (COSTS OFF) SELECT * FROM op_test WHERE col LIKE col::BPCHAR ORDER BY
col;
          QUERY PLAN
--------------------------------
 Seq Scan on op_test
   Filter: (col ~~ (col)::text)
(2 rows)
gaussdb=#  SELECT * FROM op_test WHERE col LIKE col::BPCHAR ORDER BY col;
 col
-----
 11
 12
 aa
 sd
(4 rows)
gaussdb=#  SET behavior_compat_options = 'enable_bpcharlikebpchar_compare';
SET
gaussdb=#  SHOW behavior_compat_options;
```



```
    behavior_compat_options
--------------------------------
 enable_bpcharlikebpchar_compare
(1 row)
-- After this parameter is enabled, the latest bpcharlikebpchar operator is enabled, and the returned behavior meets the expected behavior during matching.
gaussdb=#  EXPLAIN (COSTS OFF) SELECT * FROM op_test WHERE col LIKE col::BPCHAR ORDER BY col;
          QUERY PLAN
--------------------------------
 Seq Scan on op_test
   Filter: (col ~~ col)
(2 rows)
gaussdb=#  SELECT * FROM op_test WHERE col LIKE col::BPCHAR ORDER BY col;
 col
-----
 1
 11
 12
 a
 aa
 sd
(6 rows)
gaussdb=#  DROP TABLE op_test;
DROP TABLE
```

NOTE

- bpcharlikebpchar can be used only when the database compatibility parameter **SQL_COMPATIBILITY** is set to **A** and the GUC parameter **behavior_compat_options** contains the **enable_bpcharlikebpchar_compare** column.
- If this feature is enabled, the result set and execution plan for the LIKE pattern matching of the BPCHAR types are affected.
- **SET behavior_compat_options='';** indicates that this feature is disabled, and **SET behavior_compat_options='enable_bpcharlikebpchar_compare'** indicates that this feature is enabled.
- After the new feature is enabled, fixed-length matching is used (bpchar matches bpchar). The parameter lengths on the left and right sides must be the same. During pattern matching, ensure that the length of the pattern column is the same as the forcible conversion length to avoid the difference between the result and the expected result caused by filling spaces after the length is too long.

- bpcharnlikebpchar(BPCHAR, BPCHAR)

  Description: Determines whether the BPCHAR character string of the first input parameter is NOT LIKE and whether the BPCHAR character string of the second input parameter is NOT LIKE. It is used to add the BPCHAR type and the NOT LIKE operator of the BPCHAR type. To enable the !~~ operator, ensure that the value of **behavior_compat_options** contains the **enable_bpcharlikebpchar_compare** configuration item.

  Return value type: Boolean

  Value range:

  – **t**: Two parameters of the BPCHAR type are matched.

  – **f**: Two parameters of the BPCHAR type are not matched.

  Example:

```
gaussdb=#  SELECT bpcharnlikebpchar('455 '::BPCHAR(10), '455'::BPCHAR(11));
 bpcharnlikebpchar
-------------------
 t
(1 row)
```



```
gaussdb=#  SELECT bpcharnlikebpchar('455 '::BPCHAR(10), '455'::BPCHAR(10));
 bpcharnlikebpchar
-------------------
 f
(1 row)

gaussdb=#  SELECT bpcharnlikebpchar('455 '::BPCHAR(10), '455 '::BPCHAR);
 bpcharnlikebpchar
-------------------
 t
(1 row)
gaussdb=#  CREATE TABLE op_test (
   col BPCHAR(2) DEFAULT NULL
 );
CREATE TABLE
gaussdb=#  CREATE INDEX op_index ON op_test(col);
CREATE INDEX
gaussdb=#  INSERT INTO op_test VALUES ('a');
INSERT 0 1
gaussdb=#  INSERT INTO op_test VALUES ('1');
INSERT 0 1
gaussdb=# INSERT INTO op_test VALUES ('11');
INSERT 0 1
gaussdb=# insert into op_test VALUES ('12');
INSERT 0 1
gaussdb=# INSERT INTO op_test VALUES ('sd');
INSERT 0 1
gaussdb=# INSERT INTO op_test VALUES ('aa');
INSERT 0 1
gaussdb=# SHOW behavior_compat_options;
 behavior_compat_options
-------------------------
(1 row)
-- If behavior_compat_options does not contain enable_bpcharlikebpchar_compare, the latest
bpcharnlikebpchar operator is not enabled and the result set returned by the matching between
bpchars is not the same as expected (no data record is returned in normal cases).
gaussdb=# SELECT * FROM op_test WHERE col NOT LIKE col::BPCHAR ORDER BY col;
 col
-----
 1
 a
(2 rows)

gaussdb=# EXPLAIN (COSTS OFF) SELECT * FROM op_test WHERE col NOT LIKE col::BPCHAR ORDER
BY col;
               QUERY PLAN
-----------------------------------------
 Index Only Scan using op_index on op_test
   Filter: (col !~~ (col)::text)
(2 rows)
gaussdb=# SET behavior_compat_options = 'enable_bpcharlikebpchar_compare';
SET
gaussdb=# SHOW behavior_compat_options;
     behavior_compat_options
---------------------------------
 enable_bpcharlikebpchar_compare
(1 row)
-- After this parameter is enabled, the latest bpcharnlikebpchar operator is enabled, and the returned
behavior meets the expected behavior during matching.

gaussdb=# SELECT * FROM op_test WHERE col NOT LIKE col::BPCHAR ORDER BY col;
 col
-----
(0 rows)

gaussdb=# EXPLAIN (COSTS OFF) SELECT * FROM op_test WHERE col NOT LIKE col::BPCHAR ORDER
BY col;
               QUERY PLAN
-----------------------------------------
```



```
 Index Only Scan using op_index on op_test
   Filter: (col !~~ col)
(2 rows)
gaussdb=# DROP TABLE op_test;
DROP TABLE
```

NOTE

- bpcharlikebpchar can be used only when the database compatibility parameter **SQL_COMPATIBILITY** is set to **A** and the GUC parameter **behavior_compat_options** contains the **enable_bpcharlikebpchar_compare** column.
- If this feature is enabled, the result set and execution plan for the NOT LIKE pattern matching of the BPCHAR types are affected.
- **SET behavior_compat_options='';** indicates that this feature is disabled, and **SET behavior_compat_options='enable_bpcharlikebpchar_compare'** indicates that this feature is enabled.
- After the new feature is enabled, fixed-length matching is used (bpchar matches bpchar). The parameter lengths on the left and right sides must be the same. During pattern matching, ensure that the length of the pattern column is the same as the forcible conversion length to avoid the difference between the result and the expected result caused by filling spaces after the length is too long.

7.6.4 Binary String Functions and Operators

String Operators

SQL defines some string functions that use keywords, rather than commas, to separate arguments.

- octet_length(string)

  Description: Specifies the number of bytes in a binary string.

  Return type: int

  Example:

  ```
  gaussdb=# SELECT octet_length(E'jo\\000se'::bytea) AS RESULT;
   result
  --------
       5
  (1 row)
  ```

- overlay(string placing string from int [for int])

  Description: Replaces substrings.

  Return type: bytea

  Example:

  ```
  gaussdb=# SELECT overlay(E'Th\\000omas'::bytea placing E'\\002\\003'::bytea from 2 for 3) AS
  RESULT;
      result
  ----------------
   \x5402036d6173
  (1 row)
  ```

- position(substring in string)

  Description: Specifies the position of a specified substring.

  Return type: int

  Example:

  ```
  gaussdb=# SELECT position(E'\\000om'::bytea in E'Th\\000omas'::bytea) AS RESULT;
   result
  ```



```
--------
     3
(1 row)
```

- substring(string [from int] [for int])

  Description: Truncates a substring.

  Return type: bytea

  Example:

```
gaussdb=# SELECT substring(E'Th\\000omas'::bytea from 2 for 3) AS RESULT;
 result
----------
 \x68006f
(1 row)
```

- substr(string, from int [, for int])

  Description: Truncates a substring.

  Return type: bytea

  Example:

```
gaussdb=# select substr(E'Th\\000omas'::bytea,2, 3) as result;
 result
----------
 \x68006f
(1 row)
```

- trim([both] bytes from string)

  Description: Removes the longest string containing only **bytes** from the start and end of **string**.

  Return type: bytea

  Example:

```
gaussdb=# SELECT trim(E'\\000'::bytea from E'\\000Tom\\000'::bytea) AS RESULT;
 result
----------
 \x546f6d
(1 row)
```

Other Binary String Functions

GaussDB provides common syntax used for calling functions.

- btrim(string bytea,bytes bytea)

  Description: Removes the longest string containing only **bytes** from the start and end of **string**.

  Return type: bytea

  Example:

```
gaussdb=# SELECT btrim(E'\\000trim\\000'::bytea, E'\\000'::bytea) AS RESULT;
   result
------------
 \x7472696d
(1 row)
```

- get_bit(string, offset)

  Description: Extracts bits from a string.

  Return type: int

  Example:

```
gaussdb=# SELECT get_bit(E'Th\\000omas'::bytea, 45) AS RESULT;
 result
```



```
--------
      1
(1 row)
```

- get_byte(string, offset)

  Description: Extracts bytes from a string.

  Return type: int

  Example:

```
gaussdb=# SELECT get_byte(E'Th\\000omas'::bytea, 4) AS RESULT;
 result
--------
    109
(1 row)
```

- set_bit(string,offset, newvalue)

  Description: Sets bits in a string.

  Return type: bytea

  Example:

```
gaussdb=# SELECT set_bit(E'Th\\000omas'::bytea, 45, 0) AS RESULT;
      result
------------------
 \x5468006f6d4173
(1 row)
```

- set_byte(string,offset, newvalue)

  Description: Sets bytes in a string.

  Return type: bytea

  Example:

```
gaussdb=# SELECT set_byte(E'Th\\000omas'::bytea, 4, 64) AS RESULT;
      result
------------------
 \x5468006f406173
(1 row)
```

- rawcmp(raw, raw)

  Description: Specifies the raw data type comparison function.

  Parameters: raw, raw.

  Return type: integer

- raweq(raw, raw)

  Description: Specifies the raw data type comparison function.

  Parameters: raw, raw.

  Return type: Boolean

- rawge(raw, raw)

  Description: Specifies the raw data type comparison function.

  Parameters: raw, raw.

  Return type: Boolean

- rawgt(raw, raw)

  Description: Specifies the raw data type comparison function.

  Parameters: raw, raw.

  Return type: Boolean

- rawin(cstring)

Description: Specifies the raw data type parsing function.
Parameter: cstring.
Return type: bytea

- rawle(raw, raw)
  Description: Specifies the raw data type parsing function.
  Parameters: raw, raw.
  Return type: Boolean
- rawlike(raw, raw)
  Description: Specifies the raw data type parsing function.
  Parameters: raw, raw.
  Return type: Boolean
- rawlt(raw, raw)
  Description: Specifies the raw data type parsing function.
  Parameters: raw, raw.
  Return type: Boolean
- rawne(raw, raw)
  Description: Compares whether the raw types are the same.
  Parameters: raw, raw.
  Return type: Boolean
- rawnlike(raw, raw)
  Description: Checks whether the raw type matches the pattern.
  Parameters: raw, raw.
  Return type: Boolean
- rawout(bytea)
  Description: Specifies the RAW output API.
  Parameter: bytea
  Return type: cstring
- rawsend(raw)
  Description: Converts the bytea type to the binary type.
  Parameter: raw
  Return type: bytea
- rawtohex(text)
  Description: Converts the raw format to the hexadecimal format.
  Parameter: text
  Return type: text

7.6.5 Bit String Functions and Operators

Bit String Operators

Aside from the usual comparison operators, the following operators can be used. Bit string operands of **&**, **|**, and **#** must be of equal length. In case of bit shifting, the original length of the string is preserved by zero padding (if necessary).

- ||

  Description: Connects bit strings.

  Example:

```
gaussdb=# SELECT B'10001' || B'011' AS RESULT;
 result
----------
 10001011
(1 row)
```

  NOTE

  - It is recommended that a column have no more than 180 consecutive internal joins. A column with over 180 joins will be split into joined consecutive strings.

    Example: **str1||str2||str3||str4** is split into **(str1||str2)||(str3||str4)**.
  - In A-compatible mode, if bit strings contain a null string, the null string is ignored and other strings are joined. In other compatibility modes, the null string is returned.

    Take **str1||NULL||str2** as an example. **str1str2** is returned in A-compatible mode and **NULL** is returned in other compatibility modes.

- &

  Description: Specifies the AND operation between bit strings.

  Example:

```
gaussdb=# SELECT B'10001' & B'01101' AS RESULT;
 result
--------
 00001
(1 row)
```

- |

  Description: Specifies the OR operation between bit strings.

  Example:

```
gaussdb=# SELECT B'10001' | B'01101' AS RESULT;
 result
--------
 11101
(1 row)
```

- #

  Description: Specifies the OR operation between bit strings if they are inconsistent. If the same positions in the two bit strings are both 1 or 0, the position returns **0**.

  Example:

```
gaussdb=# SELECT B'10001' # B'01101' AS RESULT;
 result
--------
 11100
(1 row)
```

- ~

  Description: Specifies the NOT operation between bit strings.

  Example:

```
gaussdb=# SELECT ~B'10001'AS RESULT;
 result
----------
 01110
(1 row)
```


- <<

  Description: Shifts left in a bit string.

  Example:

```
gaussdb=# SELECT B'10001' << 3 AS RESULT;
 result
----------
 01000
(1 row)
```

- >>

  Description: Shifts right in a bit string.

  Example:

```
gaussdb=# SELECT B'10001' >> 2 AS RESULT;
 result
----------
 00100
(1 row)
```

The following SQL-standard functions work on bit strings as well as strings: length, bit_length, octet_length, position, substring, and overlay.

The following functions work on bit strings as well as binary strings: get_bit and set_bit. When working with a bit string, these functions number the first (leftmost) bit of the string as bit 0.

In addition, it is possible to convert between integral values and type bit. Example:

```
gaussdb=# SELECT 44::bit(10) AS RESULT;
   result
------------
 0000101100
(1 row)

gaussdb=# SELECT 44::bit(3) AS RESULT;
 result
--------
 100
(1 row)

gaussdb=# SELECT cast(-44 as bit(12)) AS RESULT;
    result
--------------
 111111010100
(1 row)

gaussdb=# SELECT '1110'::bit(4)::integer AS RESULT;
 result
--------
     14
(1 row)

gaussdb=# SELECT substring('10101111'::bit(8), 2);
 substring
-----------
 0101111
(1 row)
```

NOTE

Casting to just "bit" means casting to bit(1), and so will deliver only the least significant bit of the integer.

7.6.6 Pattern Matching Operators

The database provides three independent methods for implementing pattern matching: SQL LIKE operator, SIMILAR TO operator, and POSIX-style regular expressions. Besides these basic operators, functions can be used to extract or replace matching substrings and to split a string at matching locations.

- LIKE

  Description: Specifies whether the string matches the pattern string following LIKE. The LIKE expression returns true if the string matches the supplied pattern. (As expected, the NOT LIKE expression returns false if LIKE returns true, and vice versa.)

  Matching rules:

  a. This operator can succeed only when its pattern matches the entire string. If you want to match a sequence in any position within the string, the pattern must begin and end with a percent sign.

  b. An underscore (_) represents (matches) any single character. A percent sign (%) represents the wildcard character of any string.

  c. To match a literal underscore or percent sign, the respective character in pattern must be preceded by the escape character. The default escape character is one backslash but a different one can be selected by using the ESCAPE clause.

  d. To match with escape characters, enter two escape characters. For example, to write a pattern constant containing a backslash (\), you need to enter two backslashes in SQL statements.

  > NOTE
  >
  > When **standard_conforming_strings** is set to **off**, any backslashes you write in literal string constants will need to be doubled. Therefore, writing a pattern that matches a single backslash actually involves writing four backslashes in the statement (you can avoid this by selecting a different escape character with **ESCAPE** so that the backslash is no longer a special character of **LIKE**. But the backslash is still the special character of the character text analyzer, so you still need two backslashes.)
  >
  > In MySQL-compatible schema, it is also possible to select no escape character by writing **ESCAPE ''**. This effectively disables the escape mechanism, which makes it impossible to turn off the special meaning of underscore and percent signs in the schema.

  e. The keyword ILIKE can be used instead of LIKE to make the match case-insensitive.

  f. Operator **~~** is equivalent to LIKE, and operator **~~*** corresponds to ILIKE.

  Example:

```
gaussdb=# SELECT 'abc' LIKE 'abc' AS RESULT;
 result
------------
 t
(1 row)
gaussdb=# SELECT 'abc' LIKE 'a%' AS RESULT;
 result
------------
 t
(1 row)
gaussdb=# SELECT 'abc' LIKE '_b_' AS RESULT;
 result
```



```
-----------
 t
(1 row)
gaussdb=# SELECT 'abc' LIKE 'c' AS RESULT;
 result
-----------
 f
(1 row)
```

- SIMILAR TO

  Description: Returns true or false depending on whether the pattern matches the given string. It is similar to LIKE, but differs in that it uses the regular expression understanding pattern defined by the SQL standard.

  Matching rules:

  a. Similar to LIKE, this operator succeeds only when its pattern matches the entire string. If you want to match a sequence in any position within the string, the pattern must begin and end with a percent sign.
  b. An underscore (_) represents (matches) any single character. A percent sign (%) represents the wildcard character of any string.
  c. SIMILAR TO supports these pattern-matching metacharacters borrowed from POSIX-style regular expressions:

Metacharacter	Description
|	Specifies alternation (either of two alternatives).
*	Specifies repetition of the previous item zero or more times.
+	Specifies repetition of the previous item one or more times.
?	Specifies repetition of the previous item zero or one time.
{m}	Specifies repetition of the previous item exactly *m* times.
{m,}	Specifies repetition of the previous item *m* or more times.
{m,n}	Specifies repetition of the previous item at least *m* times and does not exceed *n* times.
()	Combines multiple items into a logical item.
[...]	Specifies a character class, just as in POSIX-style regular expressions.

  d. A preamble escape character disables the special meaning of any of these metacharacters. The rules for using escape characters are the same as those for LIKE.

  Regular expressions:

The **substring(string from pattern for escape)** function extracts a substring that matches an SQL regular expression pattern.

Example:

```
gaussdb=# SELECT 'abc' SIMILAR TO 'abc' AS RESULT;
 result
-----------
 t
(1 row)
gaussdb=# SELECT 'abc' SIMILAR TO 'a' AS RESULT;
 result
-----------
 f
(1 row)
gaussdb=# SELECT 'abc' SIMILAR TO '%(b|d)%' AS RESULT;
 result
-----------
 t
(1 row)
gaussdb=# SELECT 'abc' SIMILAR TO '(b|c)%'  AS RESULT;
 result
-----------
 f
(1 row)
```

- POSIX-style regular expressions

  Description: A regular expression is a collation that is an abbreviated definition of a set of strings (a regular set). If a string is a member of a regular set described by a regular expression, the string matches the regular expression. POSIX-style regular expressions provide a more powerful means for pattern matching than the LIKE and SIMILAR TO operators. **Table 7-35** lists all available operators for pattern matching using POSIX-style regular expressions.

  Table 7-35 Regular expression match operators

Operator	Description	Example
~	Matches a regular expression, which is case-sensitive.	'thomas' ~ '.*thomas.*'
~*	Matches a regular expression, which is case-insensitive.	'thomas' ~* '.*Thomas.*'
!~	Does not match a regular expression, which is case-sensitive.	'thomas' !~ '.*Thomas.*'
!~*	Does not match a regular expression, which is case-insensitive.	'thomas' !~* '.*vadim.*'

  Matching rules:

a. Unlike LIKE patterns, a regular expression is allowed to match anywhere within a string, unless the regular expression is explicitly anchored to the beginning or end of the string.

b. Besides the metacharacters mentioned above, POSIX-style regular expressions also support the following pattern matching metacharacters:

Metacharacter	Description
^	Specifies the match starting with a string.
$	Specifies the match at the end of a string.
.	Matches any single character.

Regular expressions:

POSIX-style regular expressions support the following functions:

- The **substring(string from pattern)** function provides a method for extracting a substring that matches the POSIX-style regular expression pattern.
- The **regexp_count(string text, pattern text [, position int [, flags text]])** function provides the function of obtaining the number of substrings that match the POSIX-style regular expression pattern.
- The **regexp_instr(string text, pattern text [, position int [, occurrence int [, return_opt int [, flags text]]]])** function is used to obtain the position of a substring that matches a POSIX-style regular expression pattern.
- The **regexp_substr(string text, pattern text [, position int [, occurrence int [, flags text]]])** function provides a method to extract a substring that matches a POSIX-style regular expression pattern.
- The **regexp_replace(string, pattern, replacement [,flags])** function replaces the substring that matches the POSIX-style regular expression pattern with the new text.
- The **regexp_matches(string text, pattern text [, flags text])** function returns a text array consisting of all captured substrings that match a POSIX-style regular expression pattern.
- The **regexp_split_to_table(string text, pattern text [, flags text])** function splits a string using a POSIX-style regular expression pattern as a delimiter.
- The **regexp_split_to_array(string text, pattern text [, flags text])** function behaves the same as regexp_split_to_table, except that regexp_split_to_array returns its result as an array of text.

  NOTE

  The regular expression split functions ignore zero-length matches, which occur at the beginning or end of a string or after the previous match. This is contrary to the strict definition of regular expression matching. The latter is implemented by **regexp_matches**, but the former is usually the most commonly used behavior in practice.

Example:

```
gaussdb=#  SELECT 'abc' ~ 'Abc' AS RESULT;
result
--------
 f
(1 row)
gaussdb=# SELECT 'abc' ~* 'Abc' AS RESULT;
 result
--------
 t
(1 row)
gaussdb=# SELECT 'abc' !~ 'Abc' AS RESULT;
 result
--------
 t
(1 row)
gaussdb=# SELECT 'abc'!~* 'Abc' AS RESULT;
 result
--------
 f
(1 row)
gaussdb=# SELECT 'abc' ~ '^a' AS RESULT;
 result
--------
 t
(1 row)
gaussdb=# SELECT 'abc' ~ '(b|d)'AS RESULT;
 result
--------
 t
(1 row)
gaussdb=# SELECT 'abc' ~ '^(b|c)'AS RESULT;
 result
--------
 f
(1 row)
```

Although most regular expression searches can be executed quickly, they can still be artificially processed to require any length of time and any amount of memory. It is not recommended that you accept the regular expression search pattern from the non-security pattern source. If you must do this, you are advised to add the statement timeout limit. The search with the SIMILAR TO pattern has the same security risks as the SIMILAR TO provides many capabilities that are the same as those of the POSIX-style regular expression. The LIKE search is much simpler than the other two options. Therefore, it is more secure to accept the non-secure pattern source search.

- [NOT] REGEXP/ [NOT] RLIKE

  Description: The REGEXP operator is used for regular expression matching and complies with POSIX-style regular expression matching rules. **TRUE** or **FALSE** is returned based on whether the pattern matches the given string. The following table describes the regular expression operators.

Operator Name	Description	Syntax
REGEXP	Specifies whether a string matches the regular expression.	expr REGEXP pat

Operator Name	Description	Syntax
RLIKE	Specifies whether a string matches the regular expression (same as REGEXP).	expr RLIKE pat
NOT REGEXP	Specifies whether a string does not match the regular expression.	expr NOT REGEXP pat
NOT RLIKE	Specifies whether the character string does not match the regular expression (same as NOT REGEXP).	expr NOT RLIKE pat

Matching rules:

a. A regular expression is allowed to match anywhere within a string, unless the regular expression is explicitly anchored to the beginning or end of the string.

b. The pattern matching metacharacters supported by the REGEXP operator are the same as those supported by POSIX-style regular expressions.

c. The operator supports the following escape character matching.

Escape Character	Description
\b	Backspace key.
\f	Form feed character, for example, C language.
\n	Newline character, for example, C language.
\r	Carriage return character, for example, C language.
\t	Horizontal tab, for example, C language.
\uwxyz	Character whose hexadecimal value is **0xwxyz**, where **wxyz** is four hexadecimal digits.
\xhhh	Character whose hexadecimal value is **0xhhh**, where **hhh** is any sequence of hexadecimal digits.
\0	The GaussDB reports the error **invalid byte sequence for encoding "UTF8": 0x00**.
\xy	Character whose octal value is **0xy**, where **xy** is two octal digits.

Escape Character	Description
\xyz	Character whose octal value is **0xyz**, where **xyz** is three octal digits.

d. Ranges for matching a pattern string: [a-dX] and [^a-dX].

[a-dX] matches any characters of a, b, c, d, and X. [^a-dX] matches characters other than a, b, c, d, or X.

The hyphen (-) between two characters forms a range, indicating that all characters in the range are matched. To include a right square bracket (]), it must follow the left square bracket ([). To include a hyphen (-), it must be after the left square bracket ([) or before the right square bracket (]). Any character that does not have any special meaning enclosed in square brackets ([]) matches itself.

Example:

```
gaussdb=# SELECT 'abd' REGEXP 'a[bc]d' AS RESULT;
 result
--------
 t
(1 row)
gaussdb=# SELECT 'aed' REGEXP 'a[^bc]d' AS RESULT;
 result
--------
 t
(1 row)
gaussdb=# SELECT 'a-' REGEXP 'a[-b]' AS RESULT;
 result
--------
 t
(1 row)
gaussdb=# SELECT 'aX]bc' REGEXP '^[]a-dXYZ]*$' AS RESULT;
 result
--------
 t
(1 row)
```

e. [.characters.] in the pattern string pat matches the collation of the element. In bracket expressions using quare brackets ([]), the collation used to proofread elements is matched. The character is a single character or a character name such as **space**. A complete list of character names can be found in the **regex/regc_locale** file.

Example:

```
gaussdb=# SELECT ' ' REGEXP '[[.space.]]' AS RESULT;
 result
--------
 t
(1 row)
```

f. Character class matching the **[=character_class=]** character in the pattern string pat. It is written in the square bracket expression. **[=character_class=]** indicates the equivalence class. A character matches all characters with the same sort proofreading value, including itself. For example, if o and (+) are of the same class, [[=o=]], [[=(+)=]], and [o(+)] are synonyms. The same class cannot be used as an endpoint of a range.

g. Character class matching the **[:character_class:]** character in the pattern string pat. It is written in square brackets ([]). **[:character_class:]** is used to match the characters that match the character class. Other class names may be provided for specific regions. The character class cannot be used as an endpoint of a range. The following table lists the standard class names. If the backslash (\) is involved, set the parameters according to the description.

Character Class	Description	Character Range
alnum	Alphanumeric numeric character.	[0-9a-zA-Z]
alpha	Alphanumeric character.	[a-zA-Z]
blank	Blank character.	[\t], indicating a blank character
cntrl	Control character.	[\x01-\x1F]
digit	Digit character.	[0-9]
graph	Graphic character.	[^\x01-\x20]
lower	Lowercase character.	[a-z]
print	Graphic character.	[^\x01-\x20]
punct	Punctuation character.	[-!"#$%&'( )*+,./:;<=>?@[\ \]^_`{|}~]
space	Spaces, tabs, new lines, and carriage returns.	[\n\r\t\x0B]
upper	Uppercase character.	[A-Z]
xdigit	Hexadecimal numeric character.	[0-9a-fA-F]

Example:

```
gaussdb=# SELECT '\n' REGEXP '[[:space:]]' AS RESULT;
 result
--------
 t
(1 row)
```

h. Start and end matching of the [[:<:]], [[:>:]] matching string in the pattern string pat.

Example:

```
gaussdb=# SELECT 'a word a' REGEXP '[[:<:]]word[[:>:]]' AS RESULT;
 result
--------
 t
(1 row)
```


i. To match a text instance with special characters, add two backslashes (\) before the special characters (including (,), ., ", ^, +, and ?). To match single quotation marks, you need to write **'\''** in the source string to match **'\\\''** in the pattern string. If the backslash (\) is involved, set the parameters according to the description.

Example:

```
gaussdb=# SELECT 'a+b' REGEXP 'a\\+b' AS RESULT;
 result
--------
 t
(1 row)
gaussdb=# SELECT '\'' REGEXP '\\\'' AS RESULT;
 result
--------
 t
(1 row)
gaussdb=# SELECT '\\' REGEXP '\\\\' AS RESULT;
 result
--------
 t
(1 row)
```

j. When the case-sensitive character set and collation are set, regular expression matching is also case-sensitive.

Example:

```
gaussdb=# SELECT 'abc' REGEXP 'ABC' COLLATE utf8mb4_bin AS RESULT;
 result
--------
 f
(1 row)
```

k. If the input parameter **Expr** or pat is null, **NULL** is returned.

```
gaussdb=# SELECT NULL REGEXP '*' AS RESULT;
 result
--------

(1 row)
gaussdb=# SELECT '-' REGEXP NULL AS RESULT;
 result
--------

(1 row)
```

NOTE

This operator is supported only in B-compatible databases and is valid only when **sql_compatibility** is set to **'B'**. **b_format_version** is set to **'5.7'**. In this case, the REGEXP operator is equivalent to the ~* operator, and the NOT REGEXP operator is equivalent to the !~* operator.

- When **b_format_dev_version** is set to **s2**, **standard_conforming_strings** is set to **off** and **escape_string_warning** is set to **off** by default. In this case, any backslash written in the string constant must be double-written. Therefore, writing a pattern matching a single backslash is actually going to write four backslashes in the statement.
- This operator supports only the string type, text type, and other data types that can be implicitly converted to the string type and text type. The bytea type is not supported.

7.6.7 Arithmetic Functions and Operators

Arithmetic Operators

- +

  Description: Addition

  Example:

```
gaussdb=# SELECT 2+3 AS RESULT;
 result
--------
      5
(1 row)
```

- -

  Description: Subtraction

  Example:

```
gaussdb=# SELECT 2-3 AS RESULT;
 result
--------
     -1
(1 row)
```

- *

  Description: Multiplication

  Example:

```
gaussdb=# SELECT 2*3 AS RESULT;
 result
--------
      6
(1 row)
```

- /

  Description: Division (The result is not rounded.)

  Example:

```
gaussdb=# SELECT 4/2 AS RESULT;
 result
--------
      2
(1 row)
gaussdb=# SELECT 4/3 AS RESULT;
       result
------------------
 1.33333333333333
(1 row)
```

- +/-

  Description: Positive/Negative

  Example:

```
gaussdb=# SELECT -2 AS RESULT;
 result
--------
     -2
(1 row)
```

- %

  Description: Model (to obtain the remainder)

  Example:

```
gaussdb=# SELECT 5%4 AS RESULT;
 result
```



```
--------
       1
(1 row)
```

- @

  Description: Absolute value

  Example:

```
gaussdb=# SELECT @ -5.0 AS RESULT;
 result
--------
    5.0
(1 row)
```

- ^

  Description: Power (exponent calculation)

  Example:

```
gaussdb=# SELECT 2.0^3.0 AS RESULT;
       result
--------------------
 8.0000000000000000
(1 row)
```

- |/

  Description: Square root

  Example:

```
gaussdb=# SELECT |/ 25.0 AS RESULT;
 result
--------
      5
(1 row)
```

- ||/

  Description: Cubic root

  Example:

```
gaussdb=# SELECT ||/ 27.0 AS RESULT;
 result
--------
      3
(1 row)
```

- !

  Description: Factorial

  Example:

```
gaussdb=# SELECT 5! AS RESULT;
 result
--------
    120
(1 row)
```

- !!

  Description: Factorial (prefix operator)

  Example:

```
gaussdb=# SELECT !!5 AS RESULT;
 result
--------
    120
(1 row)
```

- &

  Description: Binary AND

Example:

```
gaussdb=# SELECT 91&15  AS RESULT;
 result
--------
     11
(1 row)
```

- |

  Description: Binary OR

  Example:

  ```
  gaussdb=# SELECT 32|3  AS RESULT;
   result
  --------
     35
  (1 row)
  ```

- #

  Description: Binary XOR

  Example:

  ```
  gaussdb=# SELECT 17#5  AS RESULT;
   result
  --------
     20
  (1 row)
  ```

- ~

  Description: Binary NOT

  Example:

  ```
  gaussdb=# SELECT ~1 AS RESULT;
   result
  --------
     -2
  (1 row)
  ```

- <<

  Description: Binary shift left

  Example:

  ```
  gaussdb=# SELECT 1<<4 AS RESULT;
   result
  --------
     16
  (1 row)
  ```

- >>

  Description: Binary shift right

  Example:

  ```
  gaussdb=# SELECT 8>>2 AS RESULT;
   result
  --------
      2
  (1 row)
  ```

Arithmetic Functions

- abs(x)

  Description: Absolute value

  Return type: same as the input

  Example:


```
gaussdb=# SELECT abs(-17.4);
 abs
------
 17.4
(1 row)
```

- acos(x)

  Description: Arc cosine

  Return type: double precision

  Example:

```
gaussdb=# SELECT acos(-1);
       acos
------------------
 3.14159265358979
(1 row)
```

- asin(x)

  Description: Arc sine

  Return type: double precision

  Example:

```
gaussdb=# SELECT asin(0.5);
       asin
------------------
 .523598775598299
(1 row)
```

- atan(x)

  Description: Arc tangent

  Return type: double precision

  Example:

```
gaussdb=# SELECT atan(1);
       atan
------------------
 .785398163397448
(1 row)
```

- atan2(y, x)

  Description: Arc tangent of y/x

  Return type: double precision

  Example:

```
gaussdb=# SELECT atan2(2, 1);
      atan2
------------------
 1.10714871779409
(1 row)
```

- bitand(integer, integer)

  Description: Performs the AND (&) operation on two integers.

  Return type: bigint

  Example:

```
gaussdb=# SELECT bitand(127, 63);
 bitand
--------
     63
(1 row)
```

- cbrt(dp)

  Description: Cubic root

Return type: double precision

Example:

```
gaussdb=# SELECT cbrt(27.0);
 cbrt
------
    3
(1 row)
```

- ceil(x)

  Description: Minimum integer greater than or equal to the parameter

  Return type: integer

  Example:

  ```
  gaussdb=# SELECT ceil(-42.8);
   ceil
  ------
    -42
  (1 row)
  ```

- ceiling(dp or numeric)

  Description: Minimum integer (alias of ceil) greater than or equal to the parameter

  Return type: dp or numeric. If implicit type conversion is not considered, the return type is the same as the input type.

  Example:

  ```
  gaussdb=# SELECT ceiling(-95.3);
   ceiling
  ---------
      -95
  (1 row)
  ```

- cos(x)

  Description: Cosine

  Return type: double precision

  Example:

  ```
  gaussdb=# SELECT cos(-3.1415927);
          cos
  -------------------
   -.999999999999999
  (1 row)
  ```

- cosh(x)

  Description: Hyperbolic cosine

  Return type: dp or numeric. If implicit type conversion is not considered, the return type is the same as the input type.

  Example:

  ```
  gaussdb=# SELECT cosh(4);
          cosh
  -------------------
   27.3082328360165
  (1 row)
  ```

  NOTE

  This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an A-compatible database.

- cot(x)

  Description: Cotangent

Return type: double precision

Example:

```
gaussdb=# SELECT cot(1);
       cot
------------------
 .642092615934331
(1 row)
```

- degrees(dp)

  Description: Converts radians to angles.

  Return type: double precision

  Example:

  ```
  gaussdb=# SELECT degrees(0.5);
      degrees
  ------------------
   28.6478897565412
  (1 row)
  ```

- div(y numeric, x numeric)

  Description: Integer part of y/x

  Return type: numeric

  Example:

  ```
  gaussdb=# SELECT div(9,4);
   div
  -----
     2
  (1 row)
  ```

- exp(x)

  Description: Natural exponent

  Return type: dp or numeric. If implicit type conversion is not considered, the return type is the same as the input type.

  Example:

  ```
  gaussdb=# SELECT exp(1.0);
          exp
  --------------------
   2.7182818284590452
  (1 row)
  ```

- floor(x)

  Description: Not larger than the maximum integer of the parameter

  Return type: same as the input

  Example:

  ```
  gaussdb=# SELECT floor(-42.8);
   floor
  -------
     -43
  (1 row)
  ```

- int1(in)

  Description: Converts the input text parameter to a value of the int1 type and returns the value.

  Return type: int1

  Example:

  ```
  gaussdb=# SELECT int1('123');
   int1
  ```



```
------
 123
(1 row)
gaussdb=# SELECT int1('1.1');
 int1
------
   1
(1 row)
```

- int2(in)

  Description: Converts the input parameter to a value of the int2 type and returns the value.

  The supported input parameter types include float4, float8, int16, numeric, and text.

  Return type: int2

  Example:

```
gaussdb=# SELECT int2('1234');
 int2
------
 1234
(1 row)
gaussdb=# SELECT int2(25.3);
 int2
------
   25
(1 row)
```

- int4(in)

  Description: Converts the input parameter to a value of the int4 type and returns the value.

  The supported input parameter types include bit, boolean, char, double precision, int16, numeric, real, smallint, and text

  Return type: int4

  Example:

```
gaussdb=# SELECT int4('789');
 int4
------
 789
(1 row)
gaussdb=# SELECT int4(99.9);
 int4
------
  100
(1 row)
```

- int8(in)

  Description: Converts the input parameter to a value of the int8 type and returns the value. The supported input parameter types include bit, double precision, int16, integer, numeric, oid, real, smallint, and text.

  Return type: int8

  Example:

```
gaussdb=# SELECT int8('789');
 int8
------
 789
(1 row)
gaussdb=# SELECT int8(99.9);
 int8
------
```



```
  99
(1 row)
```

- float4(in)

  Description: Converts the input parameter to a value of the float4 type and returns the value. The supported input parameter types include bigint, double precision, int16, integer, numeric, smallint, and text.

  Return type: float4

  Example:

```
gaussdb=# SELECT float4('789');
 float4
--------
    789
(1 row)

gaussdb=# SELECT float4(99.9);
 float4
--------
   99.9
(1 row)
```

- float8(in)

  Description: Converts the input parameter to a value of the float8 type and returns the value. The supported input parameter types include bigint, int16, integer, numeric, real, smallint, and text.

  Return type: float8

  Example:

```
gaussdb=# SELECT float8('789');
 float8
--------
    789
(1 row)

gaussdb=# SELECT float8(99.9);
 float8
--------
   99.9
(1 row)
```

- int16(in)

  Description: Converts the input parameter to a value of the int16 type and returns the value. The supported input parameter types include bigint, Boolean, double precision, integer, numeric, oid, real, smallint, and tinyint.

  Return type: int16

  Example:

```
gaussdb=# SELECT int16('789');
 int16
--------
    789
(1 row)

gaussdb=# SELECT int16(99.9);
 int16
--------
   100
(1 row)
```

- numeric(in)

  Description: Converts the input parameter to a value of the numeric type and returns the value. The supported input parameter types include bigint, boolean, double precision, int16, integer, money, real, and smallint.

Return type: numeric

Example:

```
gaussdb=# SELECT "numeric"('789');
 numeric
---------
     789
(1 row)

gaussdb=# SELECT"numeric"(99.9);
 numeric
---------
    99.9
(1 row)
```

- oid(in)

  Description: Converts the input parameter to a value of the oid type and returns the value. The supported input parameter types include bigint and int16.

  Return type: oid

- radians(dp)

  Description: Converts angles to radians.

  Return type: double precision

  Example:

  ```
  gaussdb=# SELECT radians(45.0);
       radians
  ------------------
   .785398163397448
  (1 row)
  ```

- random()

  Description: Random number between 0.0 and 1.0.

  Return type: double precision

  Example:

  ```
  gaussdb=# SELECT random();
        random
  ------------------
   .824823560658842
  (1 row)
  ```

- rand([seed])

  Description: Returns a random number between 0 and 1. If seed is specified, the random value of seed is returned. There can be no input parameter or a seed input parameter of the bigint type.

  Parameter: bigint type, specifying a random number seed.

  Return type: double

  Example:

  ```
  b_compatible_db=# SELECT rand();
        rand
  ------------------
   .327476012520492
  (1 row)

  b_compatible_db=# SELECT rand(12321);
        rand
  ------------------
   .326073104515672
  (1 row)
  ```


> **NOTE**
>
> This function is valid only when **sql_compatibility** is set to **'B'**. The return value omits the trailing zeros.

- multiply(x double precision or text, y double precision or text)

  Description: product of x and y.

  Return type: double precision

  Example:

```
gaussdb=# SELECT multiply(9.0, '3.0');
    multiply
-------------------
              27
(1 row)
gaussdb=# SELECT multiply('9.0', 3.0);
    multiply
-------------------
              27
(1 row)
```

- ln(x)

  Description: Natural logarithm.

  Return type: dp or numeric. If implicit type conversion is not considered, the return type is the same as the input type.

  Example:

```
gaussdb=# SELECT ln(2.0);
         ln
-------------------
 .6931471805599453
(1 row)
```

- log(x)

  Description: Logarithm with 10 as the base.

  Return type: same as the input.

  Example:

```
gaussdb=# SELECT log(100.0);
        log
--------------------
 2.0000000000000000
(1 row)
```

- log(b numeric, x numeric)

  Description: Logarithm with b as the base.

  Return type: numeric

  Example:

```
gaussdb=# SELECT log(2.0, 64.0);
        log
--------------------
 6.0000000000000000
(1 row)
```

- log2(x)

  Description: Logarithm with 2 as the base.

  Return type: double precision.

  Example:

```
gaussdb=# SELECT log2(2);
 log2
```



```
------
   1
(1 row)
```

NOTE

This function is valid only when **sql_compatibility** is set to **'B'**.

- log10(x)

  Description: Logarithm with 10 as the base.

  Return type: double precision.

  Example:

```
gaussdb=# SELECT log10(10);
 log10
-------
     1
(1 row)
```

NOTE

This function is valid only when **sql_compatibility** is set to **'B'**.

- mod(x,y)

  Description: Remainder of x/y (model). If x equals to 0, 0 is returned.

  Return type: same as the parameter type.

  Example:

```
gaussdb=# SELECT mod(9,4);
 mod
-----
   1
(1 row)
gaussdb=# SELECT mod(9,0);
 mod
-----
   9
(1 row)
```

- pi()

  Description: π constant value.

  Return type: double precision

  Example:

```
gaussdb=# SELECT pi();
        pi
------------------
 3.14159265358979
(1 row)
```

- power(a double precision, b double precision)

  Description: b power of a.

  Return type: double precision

  Example:

```
gaussdb=# SELECT power(9.0, 3.0);
        power
----------------------
 729.0000000000000000
(1 row)
```

- remainder(x,y)

  Description: Remainder of x/y. If y is 0, an error is reported.

Return type: same as the input (float4, float8, or numeric)

Example:

```
gaussdb=# SELECT remainder(11,4);
 remainder
-----------
        -1
(1 row)
gaussdb=# SELECT remainder(9,0);
ERROR: division by zero
CONTEXT:  referenced column: remainder
```

NOTE

This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an A-compatible database.

- round(x)

  Description: Integer closest to the input parameter.

  Return type: same as the input (double precision or numeric).

  Example:

  ```
  gaussdb=# SELECT round(42.4);
   round
  -------
     42
  (1 row)

  gaussdb=# SELECT round(42.6);
   round
  -------
     43
  (1 row)
  ```

CAUTION

The output of the float/double type may be **-0**. (This also occurs in functions such as trunc and ceil. If the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s1** in an A-compatible database, the result is **0**.) The following is an example:

```
gaussdb=# SELECT round(-0.2::float8);
 round
-------
    -0
(1 row)
```

- round(v numeric, s int)

  Description: The number of *s* digits are kept after the decimal point, and the number is rounded off.

  Return type: numeric

  Example:

  ```
  gaussdb=# SELECT round(42.4382, 2);
   round
  -------
   42.44
  (1 row)
  ```


NOTE

- If the value of the control parameter *s* is a decimal, the value of *s* is truncated to an integer when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s1** in an A-compatible database. Otherwise, the value of *s* is rounded off to an integer.
- If the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s1**, the round function supports round(timestamp, text) overloading. When **(text, text)** or **(text, '')** is used as the input parameter to call the round function, round (timestamp, text) is preferred.

- setseed(dp)

  Description: Sets seed for the following random() calling (from –1.0 to 1.0, inclusive).

  Return type: void

  Example:

```
gaussdb=# SELECT setseed(0.54823);
 setseed
---------

(1 row)
```

- sign(x)

  Description: Returns symbols of this parameter.

  Return type: **–1** indicates negative numbers. **0** indicates 0, and **1** indicates positive numbers.

  Example:

```
gaussdb=# SELECT sign(-8.4);
 sign
------
   -1
(1 row)
```

- sin(x)

  Description: Sine

  Return type: double precision

  Example:

```
gaussdb=# SELECT sin(1.57079);
       sin
------------------
 .999999999979986
(1 row)
```

- sinh(x)

  Description: Hyperbolic sine

  Return type: dp or numeric. If implicit type conversion is not considered, the return type is the same as the input type.

  Example:

```
gaussdb=# SELECT sinh(4);
       sinh
------------------
 27.2899171971277
(1 row)
```

NOTE

This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an A-compatible database.

- sqrt(x)

  Description: Square root

  Return type: dp or numeric. If implicit type conversion is not considered, the return type is the same as the input type.

  Example:

```
gaussdb=# SELECT sqrt(2.0);
       sqrt
-------------------
 1.414213562373095
(1 row)
```

  NOTE

  This function uses the Karatsuba sqrt square root algorithm when the GUC parameter **gs_format_behavior_compat_options** is set to **sqrt_karatsuba**. Otherwise, this function uses the Newton iteration algorithm. The performance of the Karatsuba sqrt square root algorithm is better. In a few scenarios, the precision of the Karatsuba sqrt square root algorithm is different from that of the Newton iteration algorithm.

- tan(x)

  Description: Tangent

  Return type: double precision

  Example:

```
gaussdb=# SELECT tan(20);
       tan
------------------
 2.23716094422474
(1 row)
```

- tanh(x)

  Description: Hyperbolic tangent

  Return type: same as the input (double precision or numeric)

  Example:

```
gaussdb=# SELECT tanh(0.1);
                tanh
-----------------------------------------
0.099667994624955817118305083678352183538 9
(1 row)
```

  NOTE

  This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an A-compatible database.

- trunc(x)

  Description: Truncation (the integer part is retained).

  Return type: same as the input.

  Example:

```
gaussdb=# SELECT trunc(42.8);
 trunc
-------
    42
(1 row)
```

- trunc(v numeric, s int)

  Description: Truncates the last *s* digits after a decimal point.

  Return type: numeric

Example:

```
gaussdb=# SELECT trunc(42.4382, 2);
 trunc
-------
 42.43
(1 row)
```

NOTE

This function is valid only when **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1** in A-compatible mode. If the value of **s** is a decimal, the value is truncated instead of being rounded off.

- smgrne(a smgr, b smgr)

  Description: Compares two integers of the smgr type to check whether they are different.

  Return type: Boolean

- smgreq(a smgr, b smgr)

  Description: Compares two integers of the smgr type to check whether they are equivalent.

  Return type: Boolean

- int1abs(tinyint)

  Description: Returns the absolute value of data of the uint8 type.

  Parameter: tinyint

  Return type: tinyint

- int1and(tinyint, tinyint)

  Description: Returns the bitwise AND result of two data records of the uint8 type.

  Parameter: tinyint, tinyint

  Return type: tinyint

- int1cmp(tinyint, tinyint)

  Description: Returns the comparison result of two data records of the uint8 type. If the value of the first parameter is greater, **1** is returned. If the value of the second parameter is greater, **–1** is returned. If they are the same, **0** is returned.

  Parameter: tinyint, tinyint

  Return type: integer

- int1div(tinyint, tinyint)

  Description: Returns the result of dividing two data records of the uint8 type. The result is of the float8 type.

  Parameter: tinyint, tinyint

  Return type: tinyint

- int1eq(tinyint, tinyint)

  Description: Compares two pieces of data of the uint8 type to check whether they are the same.

  Parameter: tinyint, tinyint

  Return type: Boolean

- int1ge(tinyint, tinyint)

  Description: Determines whether the value of the first parameter is greater than or equal to the value of the second parameter in two data records of the uint8 type.

  Parameter: tinyint, tinyint

  Return type: Boolean

- int1gt(tinyint, tinyint)

  Description: Performs the greater-than operation on an unsigned 1-byte integer.

  Parameter: tinyint, tinyint

  Return type: Boolean

- int1larger(tinyint, tinyint)

  Description: Returns the maximum value of an unsigned 1-byte integer.

  Parameter: tinyint, tinyint

  Return type: tinyint

- int1le(tinyint, tinyint)

  Description: Performs a less-than or an equal-to operation on an unsigned 1-byte integer.

  Parameter: tinyint, tinyint

  Return type: Boolean

- int1lt(tinyint, tinyint)

  Description: Performs a less-than operation on an unsigned 1-byte integer.

  Parameter: tinyint, tinyint

  Return type: Boolean

- int1smaller(tinyint, tinyint)

  Description: Calculates the minimum value of an unsigned 1-byte integer.

  Parameter: tinyint, tinyint

  Return type: tinyint

- int1inc(tinyint)

  Description: Unsigned 1-byte integer plus 1.

  Parameter: tinyint

  Return type: tinyint

- int1mi(tinyint, tinyint)

  Description: Performs a minus operation on an unsigned 1-byte integer.

  Parameter: tinyint, tinyint

  Return type: tinyint

- int1mod(tinyint, tinyint)

  Description: Performs a reminder operation on an unsigned 1-byte integer.

  Parameter: tinyint, tinyint

  Return type: tinyint

- int1mul(tinyint, tinyint)

Description: Performs a multiplication operation on an unsigned 1-byte integer.

Parameter: tinyint, tinyint

Return type: tinyint

- int1ne(tinyint, tinyint)

  Description: Performs a not-equal-to operation on an unsigned 1-byte integer.

  Parameter: tinyint, tinyint

  Return type: Boolean

- int1pl(tinyint, tinyint)

  Description: Performs an addition operation on an unsigned 1-byte integer.

  Parameter: tinyint, tinyint

  Return type: tinyint

- int1um(tinyint)

  Description: Returns an unsigned 2-byte integer after subtracting the opposite number from the unsigned 1-byte integer.

  Parameter: tinyint

  Return type: smallint

- int1xor(tinyint, tinyint)

  Description: Performs an exclusive OR operation on an unsigned 1-byte integer.

  Parameter: tinyint, tinyint

  Return type: tinyint

- cash_div_int1(money, tinyint)

  Description: Performs a division operation on the money type.

  Parameter: money, tinyint

  Return type: money

- cash_mul_int1(money, tinyint)

  Description: Performs a multiplication operation on the money type.

  Parameter: money, tinyint

  Return type: money

- int1not(tinyint)

  Description: Reverts binary bits of an unsigned 1-byte integer.

  Parameter: tinyint

  Return type: tinyint

- int1or(tinyint, tinyint)

  Description: Performs an OR operation on an unsigned 1-byte integer.

  Parameter: tinyint, tinyint

  Return type: tinyint

- int1shl(tinyint, integer)

  Description: Shifts an unsigned 1-byte integer leftwards by a specified number of bits.

  Parameter: tinyint, integer

Return type: tinyint

- int1shr(tinyint, integer)

  Description: Shifts an unsigned 1-byte integer rightwards by a specified number of bits.

  Parameter: tinyint, integer

  Return type: tinyint

- width_bucket(op numeric, b1 numeric, b2 numeric, count int)

  Description: Returns a bucket to which the operand will be assigned in an equi-depth histogram with **count** buckets, ranging from **b1** to **b2**.

  Return type: int

  Example:

  ```
  gaussdb=# SELECT width_bucket(5.35, 0.024, 10.06, 5);
   width_bucket
  --------------
             3
  (1 row)
  ```

- width_bucket(op dp, b1 dp, b2 dp, count int)

  Description: Returns a bucket to which the operand will be assigned in an equi-depth histogram with **count** buckets, ranging from **b1** to **b2**.

  Return type: int

  Example:

  ```
  gaussdb=# SELECT width_bucket(5.35, 0.024, 10.06, 5);
   width_bucket
  --------------
             3
  (1 row)
  ```

- analyze_tgtype_for_type(n smallint)

  Description: Parses **pg_trigger.tgtype**, parses *n* by bit, and returns one of **before each row**, **after each row**, **before statement**, **after statement**, and **instead of**.

  Return type: varchar2(16)

- analyze_tgtype_for_event(n smallint)

  Description: Parses **pg_trigger.tgtype**, parses *n* by bit, and returns one or more of **insert**, **update**, **delete**, and **truncate**.

  Return type: varchar2(246)

- nanvl(n2, n1)

  Description: Two parameters are entered. The parameters must be of the numeric type or a non-numeric type that can be implicitly converted to the numeric type. If the first parameter **n2** is NaN, **n1** is returned. Otherwise, **n2** is returned.

  Return value type: input parameter with a higher priority. The priority is as follows: double precision > float4 > numeric.

  Example:

  ```
  gaussdb=# SELECT nanvl('NaN', 1.1);
   nanvl
  -------
    1.1
  (1 row)
  ```


> **NOTE**
>
> This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an A-compatible database.

- numeric_eq_text(numeric, text)

  Description: Checks whether the numeric variable is equal to the numeric value converted from the text variable.

  Return type: Boolean

  Example:

  ```
  gaussdb=# SELECT numeric_eq_text(1, '1');
   numeric_eq_text
  -----------------
   t
  (1 row)
  ```

- numeric_ne_text(numeric, text)

  Description: Checks whether the numeric variable is not equal to the numeric value converted from the text variable.

  Return type: Boolean

- numeric_gt_text(numeric, text)

  Description: Checks whether the numeric variable is greater than the numeric value converted from the text variable.

  Return type: Boolean

- numeric_ge_text(numeric, text)

  Description: Checks whether the numeric variable is greater than or equal to the numeric value converted from the text variable.

  Return type: Boolean

- numeric_lt_text(numeric, text)

  Description: Checks whether the numeric variable is smaller than the numeric value converted from the text variable.

  Return type: Boolean

- numeric_le_text(numeric, text)

  Description: Checks whether the numeric variable is less than or equal to the numeric value converted from the text variable.

  Return type: Boolean

- text_eq_numeric(text, numeric)

  Description: Checks whether the numeric value converted from a text variable is equal to the numeric variable.

  Return type: Boolean

  Example:

  ```
  gaussdb=# SELECT text_eq_numeric('1', 1);
   text_eq_numeric
  -----------------
   t
  (1 row)
  ```

- text_ne_numeric(text, numeric)

  Description: Checks whether the numeric value converted from a text variable is not equal to the numeric variable.

Return type: Boolean

- text_gt_numeric(text, numeric)

  Description: Checks whether the numeric value converted from a text variable is greater than the numeric variable.

  Return type: Boolean

- text_ge_numeric(text, numeric)

  Description: Checks whether the numeric value converted from a text variable is greater than or equal to the numeric variable.

  Return type: Boolean

- text_lt_numeric(text, numeric)

  Description: Checks whether the numeric value converted from a text variable is less than the numeric variable.

  Return type: Boolean

- text_le_numeric(text, numeric)

  Description: Checks whether the numeric value converted from a text variable is less than or equal to the numeric variable.

  Return type: Boolean

- bigint_eq_text(bigint, text)

  Description: Checks whether the bigint variable is equal to the bigint value converted from a text variable.

  Return type: Boolean

  ```
  gaussdb=# SELECT bigint_eq_text(1, '1');
   bigint_eq_text
  ----------------
   t
  (1 row)
  ```

- bigint_ne_text(bigint, text)

  Description: Checks whether the bigint variable is not equal to the bigint value converted from a text variable.

  Return type: Boolean

- bigint_gt_text(bigint, text)

  Description: Checks whether the bigint variable is greater than the bigint value converted from a text variable.

  Return type: Boolean

- bigint_ge_text(bigint, text)

  Description: Checks whether the bigint variable is greater than or equal to the bigint value converted from a text variable.

  Return type: Boolean

- bigint_lt_text(bigint, text)

  Description: Checks whether the bigint variable is smaller than the bigint value converted from a text variable.

  Return type: Boolean

- bigint_le_text(bigint, text)

  Description: Checks whether the bigint variable is less than or equal to the bigint value converted from a text variable.

Return type: Boolean

- text_eq_bigint(text, bigint)

  Description: Checks whether the bigint value converted from a text variable is equal to the bigint variable.

  Return type: Boolean

  Example:

```
gaussdb=# SELECT text_eq_bigint('1', 1);
 text_eq_bigint
----------------
 t
(1 row)
```

- text_ne_bigint(text, bigint)

  Description: Checks whether the bigint value converted from a text variable is not equal to the bigint variable.

  Return type: Boolean

- text_gt_bigint(text, bigint)

  Description: Checks whether the bigint value converted from a text variable is greater than the bigint variable.

  Return type: Boolean

- text_ge_bigint(text, bigint)

  Description: Checks whether the bigint value converted from a text variable is greater than or equal to the bigint variable.

  Return type: Boolean

- text_lt_bigint(text, bigint)

  Description: Checks whether the bigint value converted from a text variable is smaller than the bigint variable.

  Return type: Boolean

- text_le_bigint(text, bigint)

  Description: Checks whether the bigint value converted from a text variable is less than or equal to the bigint variable.

  Return type: Boolean

7.6.8 Date and Time Processing Functions and Operators

Date and Time Operators

Table 7-36 describes the time and date operators.

NOTE

Do not use expressions similar to 'now'::date, 'now'::timestamp,'now'::timestamptz (for string constant conversion) and text_date('now') to obtain the current database time or use the current time as the input parameter of the function. In these scenarios, the optimizer calculates the constant time in advance, causing incorrect query results.

```
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE b='now'::date;
QUERY PLAN
-----------------------------------------------------
Seq Scan on t1  (cost=0.00..13.60 rows=1 width=310)
Filter: ((b)::text = '2024-11-09 15:07:56'::text)
(2 rows)
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE b=text_date('now');
QUERY PLAN
-----------------------------------------------------
Seq Scan on t1  (cost=0.00..13.60 rows=1 width=310)
Filter: ((b)::text = '2024-11-09'::text)
(2 rows)
```

You are advised to use the now() and currenttimestamp() functions to obtain the current time of the database.

```
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE b=now();
QUERY PLAN
-----------------------------------------------------
Seq Scan on t1  (cost=0.00..14.80 rows=1 width=310)
Filter: ((b)::text = (now())::text)
(2 rows)
gaussdb=# EXPLAIN SELECT * FROM t1 WHERE b=text_date(now());
QUERY PLAN
-----------------------------------------------------------
Seq Scan on t1  (cost=0.00..16.00 rows=1 width=310)
Filter: ((b)::text = (text_date((now())::text))::text)
(2 rows)
```

When the user uses date and time operators, explicit type prefixes are modified for corresponding operands to ensure that the operands parsed by the database are consistent with what the user expects, and no unexpected results occur.

For example, abnormal mistakes will occur in the following example without an explicit data type.

```
gaussdb=# SELECT date '2001-10-01' - '7' AS RESULT;
ERROR:
GAUSS-10416: invalid input syntax for type timestamp: "7"
SQLSTATE: 22007
LINE 1: SELECT date '2001-10-01' - '7' AS RESULT;
                                 ^
CONTEXT:  referenced column: result
```

Table 7-36 Time and date operators

Operator	Example
+	gaussdb=# SELECT date '2001-9-28' + integer '7' AS RESULT; result --------------------- 2001-10-05 00:00:00 (1 row) **NOTE** In A-compatible mode, the query result is 2001-10-05 00:00:00.

Operator	Example
	gaussdb=# SELECT date '2001-09-28' + interval '1 hour' AS RESULT; result --------------------- 2001-09-28 01:00:00 (1 row)
	gaussdb=# SELECT date '2001-09-28' + time '03:00' AS RESULT; result --------------------- 2001-09-28 03:00:00 (1 row)
	gaussdb=# SELECT interval '1 day' + interval '1 hour' AS RESULT; result ---------------- 1 day 01:00:00 (1 row)
	gaussdb=# SELECT timestamp '2001-09-28 01:00' + interval '23 hours' AS RESULT; result --------------------- 2001-09-29 00:00:00 (1 row)
	gaussdb=# SELECT time '01:00' + interval '3 hours' AS RESULT; result ---------- 04:00:00 (1 row)
-	gaussdb=# SELECT date '2001-10-01' - date '2001-09-28' AS RESULT; result -------- 3days (1 row)
	gaussdb=# SELECT date '2001-10-01' - integer '7' AS RESULT; result --------------------- 2001-09-24 00:00:00 (1 row)
	gaussdb=# SELECT date '2001-09-28' - interval '1 hour' AS RESULT; result --------------------- 2001-09-27 23:00:00 (1 row)
	gaussdb=# SELECT time '05:00' - time '03:00' AS RESULT; result ---------- 02:00:00 (1 row)
	gaussdb=# SELECT time '05:00' - interval '2 hours' AS RESULT; result ---------- 03:00:00 (1 row)
	gaussdb=# SELECT timestamp '2001-09-28 23:00' - interval '23 hours' AS RESULT; result --------------------- 2001-09-28 00:00:00 (1 row)

Operator	Example
	gaussdb=# SELECT interval '1 day' - interval '1 hour' AS RESULT; result ---------- 23:00:00 (1 row)
	gaussdb=# SELECT timestamp '2001-09-29 03:00' - timestamp '2001-09-27 12:00' AS RESULT; result ---------- 1 day 15:00:00 (1 row)
*	gaussdb=# SELECT 900 * interval '1 second' AS RESULT; result ---------- 00:15:00 (1 row)
	gaussdb=# SELECT 21 * interval '1 day' AS RESULT; result --------- 21 days (1 row)
	gaussdb=# SELECT double precision '3.5' * interval '1 hour' AS RESULT; result ---------- 03:30:00 (1 row)
/	gaussdb=# SELECT interval '1 hour' / double precision '1.5' AS RESULT; result ---------- 00:40:00 (1 row)

Time and Date Functions

- age(timestamp, timestamp)

  Description: Subtracts parameters, producing a result in YYYY-MM-DD format. If the result is negative, the returned result is also negative. The input parameters may or may not contain time zones.

  Return type: interval

  Example:

  ```
  gaussdb=# SELECT age(timestamp '2001-04-10', timestamp '1957-06-13');
           age
  -------------------------
   43 years 9 mons 27 days
  (1 row)
  ```

- age(timestamp)

  Description: Subtracts the parameter value from the system time when the current SQL statement starts to be executed. The input parameter may or may not contain a time zone.

  Return type: interval

  Example:


```
gaussdb=# SELECT age(timestamp '1957-06-13');
          age
------------------------
 60 years 2 mons 18 days
(1 row)
```

- clock_timestamp()

  Description: Returns the timestamp of the system time when the current function is called. The volatile function obtains the latest timestamp for each scan. Therefore, the result of each call in a query is different.

  Return type: timestamp with time zone

  Example:

```
gaussdb=# SELECT clock_timestamp();
       clock_timestamp
-------------------------------
 2017-09-01 16:57:36.636205+08
(1 row)
```

- current_date

  Description: Returns the system date when the current SQL statement starts.

  Return type: date

  Example:

```
gaussdb=# SELECT current_date;
    date
------------
 2017-09-01
(1 row)
-- When the GUC parameter a_format_date_timestamp is enabled in A-compatible mode:
gaussdb=# SET a_format_date_timestamp=on;
SET
gaussdb=# SELECT current_date;
    current_date
---------------------
 2023-11-24 11:25:09
(1 row)
```

  NOTE

  - This function has the following behaviors when **sql_compatibility** is set to **'A'** and **a_format_date_timestamp** is set to **on**:
    - The timestamp of the system time is returned when the current SQL execution starts.
    - The return value type is timestamp without time zone. The value unit is seconds. The column name is **current_date**.
    - When **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s2**, the return value type is timestamp.
    - If the GUC parameter **a_format_date_timestamp** is disabled, the system date when the transaction starts is returned.
    - This prevents the optimizer from obtaining the constant time in advance. As a result, the obtained time is incorrect in the gplan scenario.
  - This function has the following behavior when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**:
    - current_date can be called with parentheses.
    - The actually called function of current_date is curdate. You can run the **\df curdate** command to query the detailed input parameters and return values of the function.

- current_time

Description: Specifies the system time when the current transaction starts. If **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, the system time when the current SQL execution starts is returned.

Return type: time with time zone. When **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, the return type is time without time zone.

Example:

```
gaussdb=# SELECT current_time;
      timetz
--------------------
 16:58:07.086215+08
(1 row)

-- When the parameter is enabled in B compatibility mode:
gaussdb_m=# SELECT current_time;
 current_time
--------------
 15:14:00
(1 row)
```

- current_time([precision])

  Description: Returns the system time when the current SQL execution starts.

  Parameter: **precision** indicates the precision (number of decimal places after the second). The value is of the int type and in the range [0,6]. The default value is **0**. If the precision is invalid, an error is reported.

  Return type: time without time zone

  Implementation mode: This function is mapped to the system function curtime.

  Example:

  ```
  gaussdb_m=# SELECT current_time();
   current_time
  --------------
   15:14:05
  (1 row)
  gaussdb_m=# SELECT current_time(3);
   current_time
  --------------
   15:14:08.433
  (1 row)
  ```

  NOTE

  This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**. The actually called function of current_time is curtime. You can run the **\df curtime** command to query the detailed input parameters and return values of the function.

- current_timestamp

  Description: Specifies the current date and time.

  Return type: timestamp without time zone when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, or timestamp with time zone in other scenarios

  Example:

  ```
  gaussdb=# SELECT current_timestamp;
       pg_systimestamp
  -------------------------------
   2017-09-01 16:58:19.22173+08
  ```



```
(1 row)

-- When the version is 5.7 in B-compatible database:
gaussdb_m=# SELECT current_timestamp;
     timestamp
---------------------
 2023-08-21 15:08:24
(1 row)
-- When the GUC parameter a_format_date_timestamp is enabled in A-compatible mode:
gaussdb=# SET a_format_date_timestamp=on;
SET
gaussdb=# SELECT current_timestamp;
      current_timestamp
-------------------------------
 2023-11-24 11:31:04.895312+08
(1 row)
```

NOTE

This function has the following behaviors when **sql_compatibility** is set to **'A'** and **a_format_date_timestamp** is set to **on**:

- The timestamp of the system time is returned when the current SQL execution starts.
- The return value type is timestamp with time zone, and the column name is **current_timestamp**.
- If the GUC parameter **a_format_date_timestamp** is disabled, the system time is returned.

This function has the following behavior when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**:

- The return type is timestamp without time zone.
- The precision of the returned result is 0.
- The timestamp of the system time is returned when the current SQL execution starts.
- This function is implemented through TYPE conversion and has no registered function. Therefore, you can run the **\df+** command of gsql to view the function information in other compatible modes, not the function information in version 5.7 in the B-compatible mode.

- current_timestamp()

  Description: Specifies the current date and time.

  Return type: timestamp without time zone

  Example:

```
gaussdb=# SELECT current_timestamp();
     timestamp
---------------------
 2023-08-21 14:34:30
(1 row)
```

> **NOTE**
>
> This function can be used only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**. In addition, this function has the following behavior:
>
> - The return type is timestamp without time zone.
> - The precision of the returned result is 0.
> - The timestamp of the system time is returned when the current SQL execution starts.
> - This function is implemented through TYPE conversion and has no registered function. Therefore, you can run the **\df+** command of gsql to view the function information in other modes, not the function information in version 5.7 in the B-compatible mode.

- current_timestamp(precision)

  Description: Obtains the current date and time, and rounds the microseconds of the result to the specified decimal place.

  Parameter: **precision** indicates the precision (number of decimal places after the second). The value is of the int type and in the range [0,6]. The default value is **0**. If the value is an integer greater than 6, an alarm is generated, and the maximum precision value **6** is used to output the time. If the value is invalid, an error is reported.

  Return type: timestamp without time zone when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, or timestamp with time zone in other scenarios

  Example:

```
gaussdb=# SELECT current_timestamp(1);
      timestamptz
-------------------------------
 2017-09-01 16:58:19.2+08
(1 row)

-- When the version is 5.7 in B-compatible database:
gaussdb_m=# SELECT current_timestamp(1);
      timestamp
-----------------------
 2023-08-21 15:09:35.3
(1 row)

-- When the GUC parameter a_format_date_timestamp is enabled in A-compatible mode:
gaussdb=# SET a_format_date_timestamp=on;
SET
gaussdb=# SELECT current_timestamp(6);
      current_timestamp
--------------------------------
 2023-11-24 11:35:57.268592+08
(1 row)
-- If a_format_version is set to 10c and a_format_dev_version is set to s2 in an A-compatible database, precision can be an integer of the numeric type.
gaussdb=# SET a_format_version='10c';
SET
gaussdb=# SET a_format_dev_version='s2';
SET
gaussdb=# SELECT current_timestamp(6.0);
      current_timestamp
--------------------------------
 2023-11-24 14:17:17.041117+08
(1 row)
```


NOTE

- The last digit 0 of the microsecond field is not displayed. For example, 2017-09-01 10:32:19.212000 is displayed as 2017-09-01 10:32:19.212.
- This function has the following behavior when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**:
  - The return type is timestamp without time zone.
  - The timestamp of the system time is returned when the current SQL execution starts.
  - This function is implemented through TYPE conversion and has no registered function. Therefore, you can run the **\df+** command of gsql to view the function information in other modes, not the function information in version 5.7 in the B-compatible mode.
- This function has the following behaviors when **sql_compatibility** is set to **'A'** and **a_format_date_timestamp** is set to **on**:
  - The return value type is timestamp with time zone, and the column name is **current_timestamp**.
  - The timestamp of the system time is returned when the current SQL execution starts.
  - If **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s2**, the **precision** parameter can be an integer of the numeric type. Otherwise, only the int type is supported.
  - If the GUC parameter **a_format_date_timestamp** is disabled, when the input parameter is an integer without a decimal point, the returned result is the date and time of the system where the transaction is started; when the input parameter is an integer with a decimal point, the returned result is the system time.

- pg_systimestamp()

  Description: Current date and time (start of the current statement).

  Return type: timestamp with time zone

  Example:

```
gaussdb=# SELECT pg_systimestamp();
       pg_systimestamp
-------------------------------
 2015-10-14 11:21:28.317367+08
(1 row)
```

- date_part(text, timestamp)

  Description: Obtains the value of a subdomain in date or time, for example, the year or hour. It is equivalent to **extract(field from timestamp)**.

  Timestamp types: abstime, date, interval, reltime, time with time zone, time without time zone, timestamp with time zone, timestamp without time zone

  Return type: double precision

  Example:

```
gaussdb=# SELECT date_part('hour', timestamp '2001-02-16 20:38:40');
 date_part
-----------
        20
(1 row)
```

- date_part(text, interval)

  Description: Obtains the subdomain value of the date/time value. When obtaining the month value, if the value is greater than 12, obtain the remainder after it is divided by 12. It is equivalent to **extract(field from timestamp)**.

Return type: double precision

Example:

```
gaussdb=# SELECT date_part('month', interval '2 years 3 months');
 date_part
-----------
         3
(1 row)
```

- date_trunc(text, timestamp)

  Description: Truncates to the precision specified by **text**.

  Return type: interval, timestamp with time zone, timestamp without time zone

  Example:

  ```
  gaussdb=# SELECT date_trunc('hour', timestamp  '2001-02-16 20:38:40');
      date_trunc
  ---------------------
   2001-02-16 20:00:00
  (1 row)
  ```

- trunc(timestamp)

  Description: Truncates to day by default.

  Example:

  ```
  gaussdb=# SELECT trunc(timestamp  '2001-02-16
  20:38:40');
                         trunc
  ---------------------
  2001-02-16 00:00:00
  (1 row)
  ```

- trunc(arg1, arg2)

  Description: Truncates to the precision specified by **arg2**.

  Type of **arg1**: interval, timestamp with time zone, timestamp without time zone

  Type of **arg2**: text

  Return type: interval, timestamp with time zone, timestamp without time zone

  Example:

  ```
  gaussdb=# SELECT trunc(timestamp  '2001-02-16 20:38:40',
  'hour');
                       trunc
  ---------------------
  2001-02-16 20:00:00
  (1 row)
  ```

- round(arg1, arg2)

  Description: Rounds off to the precision specified by **arg2**.

  Type of **arg1**: timestamp without time zone

  Type of **arg2**: text

  Return type: timestamp without time zone

  Example:

  ```
  gaussdb=# SELECT round(timestamp  '2001-02-16 20:38:40',
  'hour');
                       round
  ---------------------
  2001-02-16 21:00:00
  (1 row)
  ```


> **NOTE**
>
> This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s1** in an A-compatible database.

- daterange(arg1, arg2)

  Description: Obtains time boundary information. The type of **arg1** and **arg2** is **date**.

  Return type: daterange

  Example:

```
gaussdb=# SELECT daterange('2000-05-06','2000-08-08');
       daterange
-------------------------
 [2000-05-06,2000-08-08)
(1 row)
```

- daterange(arg1, arg2, text)

  Description: Obtains time boundary information. The type of **arg1** and **arg2** is **date**, and the type of **text** is **text**.

  Return type: daterange

  Example:

```
gaussdb=# SELECT daterange('2000-05-06','2000-08-08','[]');
       daterange
-------------------------
 [2000-05-06,2000-08-09)
(1 row)
```

- isfinite(date)

  Description: Checks whether a date is a finite value. If the date is a finite value, **t** is returned. Otherwise, **f** is returned.

  Return type: Boolean

  Example:

```
gaussdb=# SELECT isfinite(date '2001-02-16');
 isfinite
----------
 t
(1 row)
gaussdb=# SELECT isfinite(date 'infinity');
 isfinite
----------
 f
(1 row)
```

- isfinite(timestamp)

  Description: Checks whether a timestamp is a finite value. If the timestamp is a finite value, **t** is returned. Otherwise, **f** is returned.

  Return type: Boolean

  Example:

```
gaussdb=# SELECT isfinite(timestamp '2001-02-16 21:28:30');
 isfinite
----------
 t
(1 row)
gaussdb=# SELECT isfinite(timestamp 'infinity');
 isfinite
----------
 f
(1 row)
```


- isfinite(interval)

  Description: Checks whether the interval is a finite value. If the interval is a finite value, **t** is returned. Currently, **f** cannot be returned. If **'infinity'** is entered, an error is reported.

  Return type: Boolean

  Example:

  ```
  gaussdb=# SELECT isfinite(interval '4 hours');
   isfinite
  ----------
   t
  (1 row)
  ```

- justify_days(interval)

  Description: Adjusts intervals to 30-day time periods, which are represented as months.

  Return type: interval

  Example:

  ```
  gaussdb=# SELECT justify_days(interval '35 days');
   justify_days
  --------------
   1 mon 5 days
  (1 row)
  ```

- justify_hours(interval)

  Description: Sets the time interval in days (24 hours is one day).

  Return type: interval

  Example:

  ```
  gaussdb=# SELECT JUSTIFY_HOURS(INTERVAL '27 HOURS');
   justify_hours
  ---------------
   1 day 03:00:00
  (1 row)
  ```

- justify_interval(interval)

  Description: Adjusts **interval** using **justify_days** and **justify_hours**.

  Return type: interval

  Example:

  ```
  gaussdb=# SELECT JUSTIFY_INTERVAL(INTERVAL '1 MON -1 HOUR');
   justify_interval
  ------------------
   29 days 23:00:00
  (1 row)
  ```

- localtime

  Description: Specifies the system time when the current transaction starts. If **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, the system date and time when the current SQL query execution starts is returned.

  Return type: time. If **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, a timestamp without time zone is returned.

  Example:

  ```
  gaussdb=# SELECT localtime AS RESULT;
      result
  ----------------
  ```



```
 16:05:55.664681
(1 row)

-- When the compatibility parameter is enabled and set to 'B':
gaussdb_m=# SELECT localtime;
     localtime
---------------------
 2023-08-21 15:21:57
(1 row)
```

- localtime([precision])

  Description: Returns the system date and time when the current SQL query execution starts.

  Parameter: **precision** indicates the precision (number of decimal places after the second). The value is of the int type and in the range [0,6]. The default value is **0**. If the value is invalid, an error is reported.

  Return type: timestamp without time zone

  Implementation method: Register the system function localtime.

  Example:

```
gaussdb_m=# SELECT localtime();
     localtime
---------------------
 2023-08-21 15:23:49
(1 row)
gaussdb_m=# SELECT localtime(3);
       localtime
-------------------------
 2023-08-21 15:23:51.965
(1 row)
```

  NOTE

  This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- localtimestamp

  Description: Specifies the current date and time.

  Return type: timestamp

  Example:

```
gaussdb=# SELECT localtimestamp;
        timestamp
----------------------------
 2017-09-01 17:03:30.781902
(1 row)

-- When the parameter is enabled in B-compatible mode:
gaussdb_m=# SELECT localtimestamp;
     timestamp
---------------------
 2023-08-21 15:27:55
(1 row)

-- When the GUC parameter a_format_date_timestamp is enabled in A-compatible mode:
gaussdb=# SET a_format_date_timestamp=on;
SET
gaussdb=# SELECT localtimestamp;
      localtimestamp
----------------------------
 2023-11-24 11:38:25.633231
(1 row)
```


NOTE

This function has the following behavior when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**:

- The system date and time are returned when the current SQL query execution starts.
- The return type is timestamp without time zone, and the column name is **timestamp**.

This function has the following behaviors when **sql_compatibility** is set to **'A'** and **a_format_date_timestamp** is set to **on**:

- The return value type is timestamp without time zone, and the column name is **localtimestamp**.
- The timestamp of the system time is returned when the current SQL execution starts.
- If the GUC parameter **a_format_date_timestamp** is disabled, the system date and time when the transaction is started is returned.

- localtimestamp([precision])

  Description: Specifies the current date and time.

  Parameter: **precision** indicates the precision (number of decimal places after the second). The value is of the int type and in the range [0,6]. The default value is **0**. If the value is an integer greater than 6, an alarm is generated, and the maximum precision value **6** is used to output the time. If the value is invalid, an error is reported.

  Return type: timestamp without time zone

  Example:

```
-- Calls with parentheses and without input parameters are supported only in B-compatible mode.
gaussdb_m=# SELECT localtimestamp();
     timestamp
---------------------
 2023-08-21 15:27:59
(1 row)
gaussdb_m=# SELECT localtimestamp(3);
       timestamp
-------------------------
 2023-08-21 15:28:02.445
(1 row)

-- When the GUC parameter a_format_date_timestamp is enabled in A-compatible mode:
gaussdb=# SET a_format_date_timestamp=on;
SET
gaussdb=# SELECT localtimestamp(6);
      localtimestamp
----------------------------
 2023-11-24 11:41:14.086227
(1 row)
-- If a_format_version is set to 10c and a_format_dev_version is set to s2 in an A-compatible database, precision can be an integer of the numeric type.
gaussdb=# SET a_format_version='10c';
SET
gaussdb=# SET a_format_dev_version='s2';
SET
gaussdb=# SELECT localtimestamp(6.0);
      localtimestamp
----------------------------
 2023-11-24 11:42:45.642167
(1 row)
```


NOTE

- Zeros at the end of microseconds are not displayed. For example, 2017-09-01 10:32:19.212000 is displayed as 2017-09-01 10:32:19.212.
- When **sql_compatibility** is set to **'B'**, **b_format_version** is set to **5.7**, and **b_format_dev_version** is set to **s1**, this function returns the system date and time when the current SQL execution starts. The function can be called with parentheses without input parameters.
- This function has the following behaviors when **sql_compatibility** is set to **'A'** and **a_format_date_timestamp** is set to **on**:
  - The timestamp of the system time is returned when the current SQL execution starts.
  - The return value type is timestamp without time zone, and the column name is **localtimestamp**.
  - If **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s2**, the **precision** parameter can be an integer of the numeric type. Otherwise, only the int type is supported.
  - If the GUC parameter **a_format_date_timestamp** is disabled, the system date and time when the transaction is started is returned.

- maketime(hour, minute, second)

  Description: Generates the time (of the time type) based on the input parameters hour, minute, and second. The three input parameters are of the bigint, bigint, and numeric types, respectively.

  Return type: time

  Example:

```
gaussdb=# SELECT maketime(8, 15, 26.53);
  maketime
-------------
 08:15:26.53
(1 row)

gaussdb=# SELECT maketime(-888, 15, 26.53);
  maketime
------------
 -838:59:59
(1 row)
```

  NOTE

  This function can be used only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**. In addition, this function has the following behavior:

  - The function returns **NULL** if any of the following conditions is met:
    - The value of the input parameter **minute** is less than **0** or greater than or equal to **60**.
    - The value of the input parameter **second** is less than **0** or greater than or equal to **60**.
    - The value of any parameter is **NULL**.
  - The returned result of the time type contains six decimal places. If the value of **second** contains more than six decimal places, the value is rounded off.
  - The returned value of the time type is in the range [–838:59:59,838:59:59]. If the value is out of the range, the specified boundary value is returned based on the positive and negative values of **hour**.
  - **maketime** does not support self-nesting.

- now()

Description: Returns the system date and time when the current transaction starts. The results returned in the same transaction are the same. If **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, the system date and time when the current SQL query execution starts is returned.

Return type: timestamp with time zone. If **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, a timestamp without time zone is returned.

Example:

```
gaussdb=# SELECT now();
              now
-------------------------------
 2017-09-01 17:03:42.549426+08
(1 row)
-- When the compatibility parameter is enabled and set to 'B':
gaussdb_m=# SELECT now();
      timestamp
---------------------
 2023-08-21 17:17:42
(1 row)
```

- now(precision)

  Description: Returns the system date and time when the current SQL query execution starts.

  Parameter: **precision** indicates the precision (number of decimal places after the second). The value is of the int type and in the range [0,6]. The default value is **0**. If the value is an integer greater than 6, an alarm is generated, and the maximum precision value **6** is used to output the time. If the value is invalid, an error is reported.

  Return type: timestamp without time zone

  Implementation mode: Obtain the value using a 'now' :: text :: timestamp without time zone expression.

  Example:

  ```
  gaussdb_m=# SELECT now(3);
        timestamp
  -------------------------
   2023-08-21 17:17:48.819
  (1 row)
  ```

  NOTE

  This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- timenow()

  Description: Returns the system date and time when the current SQL query execution starts.

  Return type: abstime

  Example:

  ```
  gaussdb=# SELECT timenow();
        timenow
  ------------------------
   2020-06-23 20:36:56+08
  (1 row)
  ```

- dbtimezone

  Description: Time zone of the current database.

Return type: text

Example:

```
gaussdb=# SELECT dbtimezone;
      dbtimezone
------------------------
 PRC
(1 row)
```

NOTE

This function is valid only when **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s2**.

- numtodsinterval(num, interval_unit)

  Description: Converts a number to the interval type. **num** is a numeric-typed number. **interval_unit** is a string in the following format: 'DAY' | 'HOUR' | 'MINUTE' | 'SECOND'

  You can set the GUC parameter **IntervalStyle** to **a** to be compatible with the interval output format of the function.

  Return type: interval

  Example:

```
gaussdb=# SELECT numtodsinterval(100, 'HOUR');
 numtodsinterval
-----------------
 100:00:00
(1 row)

gaussdb=# SET intervalstyle = a;
SET
gaussdb=# SELECT numtodsinterval(100, 'HOUR');
      numtodsinterval
------------------------------
 +000000004 04:00:00.000000000
(1 row)
```

NOTE

When **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s2**, an error is reported if **interval_unit** is set to **'DAY'** and **num** is set to a value greater than **1000000000**.

- numtoyminterval(num, interval_unit)

  Description: Converts a number to the interval type. **num** is a number of the numeric type, and **interval_unit** is a string of the fixed format ('YEAR'|'MONTH').

  You can set the GUC parameter **IntervalStyle** to **a** to be compatible with the interval output format of the function.

  Return type: interval

  Example:

```
gaussdb=# SELECT numtoyminterval(100, 'MONTH');
 numtoyminterval
-----------------
 8 years 4 mons
(1 row)

gaussdb=# SET intervalstyle = oracle;
SET
gaussdb=# SELECT numtodsinterval(100, 'MONTH');
 numtoyminterval
-----------------
```



```
 8-4
(1 row)
```

> **NOTE**
>
> This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an A-compatible database.

- new_time(date, timezone1,timezone2)

  Description: Returns the date and time of the time zone specified by **timezone2** when the date and time of the time zone specified by **timezone1** are date.

  Return type: timestamp

  Example:

```
gaussdb=# SELECT new_time('1997-10-10','AST','EST');
      new_time
---------------------
 1997-10-09 23:00:00
(1 row)
gaussdb=# SELECT NEW_TIME(TO_TIMESTAMP ('10-Sep-02 14:10:10.123000','DD-Mon-RR
HH24:MI:SS.FF'), 'AST', 'PST');
        new_time
-------------------------
 2002-09-10 10:10:10.123
(1 row)
```

> **NOTE**
>
> This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an A-compatible database.

- sessiontimezone()

  Description: Returns the time zone of the current session. There is no input parameter.

  Return type: text

  Example:

```
gaussdb=# SELECT SESSIONTIMEZONE;
 session_time_zone
-------------------
 PST8PDT
(1 row)
gaussdb=# SELECT LOWER(SESSIONTIMEZONE);
   lower
-----------
 @ 8 hours
(1 row)
```


> **NOTE**
>
> This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an A-compatible database.
>
> When the value of **set session time zone** is in the GMT+08:00/GMT-08:00 format, the verification fails and an error is reported. This behavior meets the expectation. If the value is **'s2'** and the "ERROR:invalid value for parameter "TimeZone" :"GMT-08:00"" error is reported when you use JDBC to create a connection, the application where the driver is located sends the same time zone parameter in GMT format to GaussDB. You can use either of the following methods to solve the problem:
>
> Method 1: Adjust the time zone of the OS on the application side and set the local time zone to the region format, for example, Asia/Shanghai.
>
> Method 2: Use the JDBC driver that matches the version on the application side. The JDBC driver changes the GMT time zone to a time zone format that can be identified by the database.

- sys_extract_utc(timestamp| timestamptz)

  Description: Extracts Coordinated Universal Time (UTC, also formerly known as Greenwich Mean Time) from a date-time value with a time zone offset or time zone region name. If no time zone is specified, the date and time are associated with the session time zone. The input parameter can be in timestamp or timestamptz format.

  Return type: timestamp

  Example:

```
gaussdb=# SELECT SYS_EXTRACT_UTC(TIMESTAMP '2000-03-28 11:30:00.00');
   sys_extract_utc
---------------------
 2000-03-28 03:30:00
(1 row)
gaussdb=# SELECT SYS_EXTRACT_UTC(TIMESTAMPTZ '2000-03-28 11:30:00.00 -08:00');
   sys_extract_utc
---------------------
 2000-03-28 19:30:00
(1 row)
```

  > **NOTE**
  >
  > This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an A-compatible database.

- tz_offset('time_zone_name' | '(+/-)hh:mi' | SESSIONTIMEZONE | DBTIMEZONE)

  Description: Returns the UTC offset of the time zone indicated by the input parameter. The input parameter has the preceding four formats.

  Return type: text

  Example:

```
gaussdb=# SELECT TZ_OFFSET('US/Pacific');
 tz_offset
-----------
 -08:00
(1 row)
gaussdb=# SELECT TZ_OFFSET(sessiontimezone);
 tz_offset
-----------
 +08:00
(1 row)
```


> **NOTE**
>
> This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2** in an A-compatible database.

- pg_sleep(seconds)

  Description: Specifies the delay time of the server thread in unit of second.

  Return type: void

  Example:

```
gaussdb=# SELECT pg_sleep(10);
 pg_sleep
----------

(1 row)
```

- statement_timestamp()

  Description: Current date and time (start of the current statement).

  Return type: timestamp with time zone

  Example:

```
gaussdb=# SELECT statement_timestamp();
      statement_timestamp
-------------------------------
 2017-09-01 17:04:39.119267+08
(1 row)
```

- sysdate

  Description: Returns the system date and time when the current SQL statement is executed.

  Return type: timestamp

  Example:

```
gaussdb=# SELECT sysdate;
       sysdate
---------------------
 2017-09-01 17:04:49
(1 row)
```

- sysdate([precision])

  Description: Returns the system date and time when a function is executed.

  Parameter: Indicates the time precision. The value is of the int type and in the range [0,6]. The default value is **0**.

  Return type: timestamp without time zone

  Example:

```
gaussdb_m=# SELECT sysdate();
     sysdate()
---------------------
 2023-08-21 17:17:42
(1 row)
gaussdb_m=# SELECT sysdate(3);
       sysdate(3)
-------------------------
 2023-08-21 17:17:48.819
(1 row)
```

> **NOTE**
>
> This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- current_sysdate()

  Description: Returns the system date and time when the current SQL query execution starts.

  Return type: timestamp

  Example:

```
gaussdb=# SELECT current_sysdate();
   current_sysdate
---------------------
 2023-06-20 20:09:02
(1 row)
```

- timeofday()

  Description: Returns the timestamp (such as **clock_timestamp**, but the return type is text) of the system time when the current function is called.

  Return type: text

  Example:

```
gaussdb=# SELECT timeofday();
              timeofday
--------------------------------------
 Fri Sep 01 17:05:01.167506 2017 CST
(1 row)
```

- transaction_timestamp()

  Description: Specifies the system date and time when the current transaction starts.

  Return type: timestamp with time zone. If **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, a timestamp without time zone is returned.

  Example:

```
gaussdb=# SELECT transaction_timestamp();
     transaction_timestamp
-------------------------------
 2017-09-01 17:05:13.534454+08
(1 row)
-- When the compatibility parameter is enabled and set to 'B':
gaussdb=# SELECT transaction_timestamp();
   transaction_timestamp
----------------------------
 2023-09-07 09:32:09.728998
(1 row)
```

- add_months(d,n)

  Description: Returns the time point *d* plus *n* months.

  d: indicates the value of the timestamp type and the value that can be implicitly converted to the timestamp type.

  n: indicates the value of the INTEGER type and the value that can be implicitly converted to the INTEGER type.

  Return type: timestamp

  Example:

```
gaussdb=# SELECT add_months(to_date('2017-5-29', 'yyyy-mm-dd'), 11) FROM sys_dummy;
     add_months
---------------------
 2018-04-29 00:00:00
(1 row)
```


> **NOTE**
>
> In the scenario where this function is in an A-compatible database, the value of **a_format_version** is **10c**, and that of **a_format_dev_version** is **s1**:
>
> - If the calculation result is greater than 9999, an error is reported.
> - If the value of *n* is a decimal, the value is truncated instead of being rounded off.

- last_day(d)

  Description: Returns the date of the last day of the month that contains *d*.

  Return type: timestamp

  Example:

```
gaussdb=# SELECT last_day(to_date('2017-01-01', 'YYYY-MM-DD')) AS cal_result;
     cal_result
---------------------
 2017-01-31 00:00:00
(1 row)
```

  > **NOTE**
  >
  > When **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, the last_day function calls the built-in function b_db_last_day. The input parameter type can be TEXT, DATE, DATETIME, or TIME. The return value is of the date type and can be a number in the datetime format.

- months_between(d1, d2)

  Description: Calculates the month difference between time points **d1** and **d2**. If both dates are the end of a month or are the same day, an integer is returned. Otherwise, the return value is a decimal and is calculated as 31 days per month.

  Return type: numeric

  Example:

```
gaussdb=# SELECT months_between(to_date('2022-10-31', 'yyyy-mm-dd'), to_date('2022-09-30',
'yyyy-mm-dd'));
 months_between
----------------
              1
(1 row)

gaussdb=# SELECT months_between(to_date('2022-10-30', 'yyyy-mm-dd'), to_date('2022-09-30',
'yyyy-mm-dd'));
 months_between
----------------
              1
(1 row)

gaussdb=# SELECT months_between(to_date('2022-10-29', 'yyyy-mm-dd'), to_date('2022-09-30',
'yyyy-mm-dd'));
    months_between
-----------------------
 .96774193548387096774
(1 row)
```

  > **NOTE**
  >
  > This function is valid only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s1** in an A-compatible database.

- next_day(x,y)

  Description: Calculates the time of the next week y started from x.

  Return type: timestamp

Example:

```
gaussdb=# SELECT next_day(timestamp '2017-05-25 00:00:00','Sunday')AS cal_result;
     cal_result
---------------------
 2017-05-28 00:00:00
(1 row)
```

- tinterval(abstime, abstime)

  Description: Creates a time interval with two pieces of absolute time.

  Return type: tinterval

  Example:

  ```
  gaussdb=# CALL tinterval(abstime 'May 10, 1947 23:59:12', abstime 'Mon May  1 00:30:30 1995');
                        tinterval
  ----------------------------------------------------------
   ["1947-05-10 23:59:12+09" "1995-05-01 00:30:30+08"]
  (1 row)
  ```

- tintervalend(tinterval)

  Description: Returns the end time of **tinterval**.

  Return type: abstime

  Example:

  ```
  gaussdb=# SELECT tintervalend('["Sep 4, 1983 23:59:12" "Oct4, 1983 23:59:12"]');
       tintervalend
  ------------------------
   1983-10-04 23:59:12+08
  (1 row)
  ```

- tintervalrel(tinterval)

  Return type: tinterval

  Return type: reltime

  Example:

  ```
  gaussdb=# SELECT tintervalrel('["Sep 4, 1983 23:59:12" "Oct4, 1983 23:59:12"]');
   tintervalrel
  --------------
   1 mon
  (1 row)
  ```

- smalldatetime_ge(smalldatetime, smalldatetime)

  Description: Checks whether the value of the first parameter is greater than or equal to that of the second parameter.

  Parameter: smalldatetime, smalldatetime

  Return type: Boolean

- smalldatetime_cmp(smalldatetime, smalldatetime)

  Description: Compares two smalldatetime values to check whether they are the same.

  Parameter: smalldatetime, smalldatetime

  Return type: integer

- smalldatetime_eq(smalldatetime, smalldatetime)

  Description: Compares two smalldatetime values to check whether they are the same.

  Parameter: smalldatetime, smalldatetime

  Return type: Boolean

- smalldatetime_gt(smalldatetime, smalldatetime)

  Description: Determines whether the first parameter is greater than the second parameter.

  Parameter: smalldatetime, smalldatetime

  Return type: Boolean

- smalldatetime_hash(smalldatetime)

  Description: Calculates the hash value corresponding to a timestamp.

  Parameter: smalldatetime

  Return type: integer

- smalldatetime_in(cstring, oid, integer)

  Description: Inputs a timestamp.

  Parameter: cstring, oid, integer

  Return type: smalldatetime

- smalldatetime_larger(smalldatetime, smalldatetime)

  Description: Returns a larger timestamp.

  Parameter: smalldatetime, smalldatetime

  Return type: smalldatetime

- smalldatetime_le(smalldatetime, smalldatetime)

  Description: Checks whether the value of the first parameter is less than or equal to that of the second parameter.

  Parameter: smalldatetime, smalldatetime

  Return type: Boolean

- smalldatetime_lt(smalldatetime, smalldatetime)

  Description: Determines whether the first parameter is less than the second parameter.

  Parameter: smalldatetime, smalldatetime

  Return type: Boolean

- smalldatetime_ne(smalldatetime, smalldatetime)

  Description: Compares two timestamps to check whether they are different.

  Parameter: smalldatetime, smalldatetime

  Return type: Boolean

- smalldatetime_out(smalldatetime)

  Description: Converts a timestamp into the external form.

  Parameter: smalldatetime

  Return type: cstring

- smalldatetime_send(smalldatetime)

  Description: Converts a timestamp to the binary format.

  Parameter: smalldatetime

  Return type: bytea

- smalldatetime_smaller(smalldatetime, smalldatetime)
  Description: Returns a smaller smalldatetime.
  Parameter: smalldatetime, smalldatetime
  Return type: smalldatetime
- smalldatetime_to_abstime(smalldatetime)
  Description: Converts smalldatetime to abstime.
  Parameter: smalldatetime
  Return type: abstime
- smalldatetime_to_time(smalldatetime)
  Description: Converts smalldatetime to time.
  Parameter: smalldatetime
  Return type: time without time zone
- smalldatetime_to_timestamp(smalldatetime)
  Description: Converts smalldatetime to timestamp.
  Parameter: smalldatetime
  Return type: timestamp without time zone
- smalldatetime_to_timestamptz(smalldatetime)
  Description: Converts smalldatetime to timestamptz.
  Parameter: smalldatetime
  Return type: timestamp with time zone
- smalldatetime_to_varchar2(smalldatetime)
  Description: Converts smalldatetime to varchar2.
  Parameter: smalldatetime
  Return type: character varying

NOTE

There are multiple methods for obtaining the current time. Select an appropriate API based on the actual service scenario.

1. The following APIs return values based on the start time of the current transaction:

```
CURRENT_DATE
CURRENT_TIME
CURRENT_TIME(precision)
CURRENT_TIMESTAMP(precision)
LOCALTIME
LOCALTIMESTAMP
LOCALTIME(precision)
LOCALTIMESTAMP(precision)
transaction_timestamp()
now()
```

   The values transferred by **CURRENT_TIME** and **CURRENT_TIMESTAMP(precision)** contain time zone information. The values transferred by **LOCALTIME** and **LOCALTIMESTAMP** do not contain time zone information. CURRENT_TIME, LOCALTIME, and LOCALTIMESTAMP can specify a precision parameter, which rounds the seconds field of the result to the decimal place. If there is no precision parameter, the result is given the full precision that can be obtained.

   Because these functions all return results by the start time of the current transaction, their values do not change throughout the transaction. This is a feature with the purpose to allow a transaction to have a consistent concept at the "current" time, so that multiple modifications in the same transaction can maintain the same timestamp. **transaction_timestamp()** is equivalent to **CURRENT_TIMESTAMP(precision)**, indicating the start time of the transaction where the current statement is located. **now()** is equivalent to **transaction_timestamp()**.

   When a transaction starts, **sql_compatibility** is set to **'A'**, and **a_format_date_timestamp** is set to **on**, the results returned by the CURRENT_DATE, CURRENT_TIMESTAMP(precision), LOCALTIMESTAMP and LOCALTIMESTAMP (precision) functions are the timestamp when the current SQL statement is started. If the GUC parameter **a_format_date_timestamp** is disabled, the returned result is the transaction start date or date and time.

   When **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, CURDATE, CURRENT_DATE, CURTIME, CURRENT_TIME, CURRENT_TIMESTAMP, NOW, LOCALTIME, LOCALTIMESTAMP, UTC_TIME and UTC_TIMESTAMP all return the start time of SQL statement execution, and SYSDATE returns the start time of function calling. transaction_timestamp() still indicates the transaction start time. The return value type is changed to timestamp without time zone.

2. The following APIs return the start time of the current statement:

```
statement_timestamp()
```

   statement_timestamp() returns the start time of the current statement (more accurately, the time when the last instruction is received from the client). The return values of **statement_timestamp()** and **transaction_timestamp()** are the same during the execution of the first instruction of a transaction, but may be different in subsequent instructions.

3. The following APIs return the actual current time when the function is called:

```
clock_timestamp()
timeofday()
```

   clock_timestamp() returns the actual current time, and its value changes even in the same SQL statement. Similar to **clock_timestamp()**, **timeofday()** also returns the actual current time. However, the result of **timeofday()** is a formatted text string instead of a timestamp with time zone information.

- convert_tz(dt, from_tz, to_tz)

  Description: Converts the date and time value **dt** from the **from_tz** time zone to the **to_tz** time zone.

Parameters: For details, see **Table 2 Parameters**.

Table 7-37 Parameters

Parameter	Type	Description	Value Range
dt	datetime, date, text, and numeric	Time and date value.	[0000-01-01 00:00:00.000000,9999-12-31 23:59:59.999999]. The maximum precision value is **6**.
from_tz /to_tz	A character string in the format of ±hh:mm	Offset compared with the UTC time, for example, '+08:00'.	[–15:59,15:00]
	Named time zone	For example, 'MET' and 'UTC'.	For details, see the **PG_TIMEZONE_NAMES** system view.

Return type: datetime

Example:

```
---- Create a B-compatible database.
gaussdb=# CREATE USER JIM PASSWORD '*********';
CREATE ROLE
gaussdb=# CREATE DATABASE testdb3 OWNER JIM DBCOMPATIBILITY = 'B';
CREATE DATABASE
--- Switch to the B-compatible database testdb3 and set session parameters.
gaussdb=#  \c testdb3
testdb3=# SET b_format_dev_version = 's1';
SET
testdb3=# SET b_format_version = '5.7';
SET

testdb3=# SELECT convert_tz(cast('2023-01-01 10:10:10' as datetime), '+00:00', '+01:00');
     convert_tz
---------------------
 2023-01-01 11:10:10
(1 row)
testdb3=# SELECT convert_tz(cast('2023-01-01' as date), '+00:00', '+01:00');
     convert_tz
---------------------
 2023-01-01 01:00:00
(1 row)
testdb3=# SELECT convert_tz('2023-01-01 10:10:10', '+00:00', '+01:00');
     convert_tz
---------------------
 2023-01-01 11:10:10
(1 row)
testdb3=# SELECT convert_tz('2023-01-01', '+00:00', '+01:00');
     convert_tz
---------------------
 2023-01-01 01:00:00
(1 row)
testdb3=# SELECT convert_tz(20230101101010, '+00:00', '+01:00');
     convert_tz
---------------------
```



```
 2023-01-01 11:10:10
(1 row)
testdb3=# SELECT convert_tz(20230101, '+00:00', '+01:00');
    convert_tz
---------------------
 2023-01-01 01:00:00
(1 row)
testdb3=# SELECT convert_tz('2023-01-01 10:10:10', 'UTC', 'PRC');
    convert_tz
---------------------
 2023-01-01 18:10:10
(1 row)
```

NOTE

This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- sec_to_time(seconds)

  Description: Converts the number of seconds into a time of the time type.

  Parameters: For details, see **Table 3 Parameters**.

Table 7-38 Parameters

Parameter	Type	Description	Value Range
seconds	numeric and text	Number of seconds	[–3020399,+3020399], corresponding to the range of the time type [–838:59:59,838:59:59]. An out-of-bounds input will be truncated to the corresponding boundary value.

Return type: time without time zone

Example:

```
---- Create a B-compatible database.
gaussdb=# CREATE USER JIM PASSWORD '*********';
CREATE ROLE
gaussdb=# CREATE DATABASE testdb3 OWNER JIM DBCOMPATIBILITY = 'B';
CREATE DATABASE
--- Switch to the B-compatible database testdb3 and set session parameters.
gaussdb=#  \c testdb3
testdb3=# SET b_format_dev_version = 's1';
SET
testdb3=# SET b_format_version = '5.7';
SET

testdb3=# SELECT sec_to_time(2000);
 sec_to_time
-------------
 00:33:20
(1 row)
testdb3=# SELECT sec_to_time('-2000');
 sec_to_time
-------------
 -00:33:20
(1 row)
```


NOTE

This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- adddate(date, INTERVAL val unit)

  Description: Returns a new date by adding a certain interval to a date.

  Parameters: For details, see **Table 4 Parameters**.

  Table 7-39 Parameters

Parameter	Type	Description	Value Range
date	Expression of the time type, text, datetime, date, or time	Date to be added with an interval.	See the value ranges of the corresponding types.
val	Integer, floating-point number, or character string	Interval to be added.	See the value ranges of the corresponding types.
unit	Keyword	Unit of the interval.	YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE, SECOND, and MICROSECOND. For details, see **Time Interval Expressions**.

  Return type: TEXT, DATE, DATETIME, or TIME WITHOUT TIMEZONE

  Example:

```
gaussdb=# SELECT ADDDATE('2018-05-01', INTERVAL 1 DAY);
  adddate
------------
 2018-05-02
(1 row)
```

  NOTE

  1. This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

  2. In the prepare statement, the second parameter of **adddate** is an interval expression. When parameter **$1** is used to completely replace **adddate**, the result is unexpected, for example, prepare p1 as select adddate('2023-01-01 10:00:00', $1); execute p1(interval 13 hour). The unexpected result returned in this test case is **'2023-01-01 10:00:00'**.

- adddate(expr, days)

  Description: Returns a new date by adding a certain number of days to a date.

  Parameters:

  - **expr**: specifies the start date and time. The value type can be expression of the time type, TEXT, DATE, DATETIME, or TIME.
  - **days**: specifies the number of days to be added. The value is of the int type.

  Return type: TEXT, DATE, DATETIME, or TIME WITHOUT TIMEZONE

  Example:

```
gaussdb=# SELECT ADDDATE('2018-05-01', 1);
  adddate
------------
 2018-05-02
(1 row)
```

  NOTE

  This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- curdate()

  Description: Returns the system date when the local function calling starts. The time zone can be changed within the same connection. The returned date is affected by the time zone.

  Return type: date

  Example:

```
gaussdb=# SELECT curdate();
  curdate
------------
 2023-08-10
(1 row)
```

  NOTE

  This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- curtime([precision])

  Description: Returns the system time when the SQL query call starts.

  Parameter: **precision** indicates the precision (number of decimal places after the second). The value is of the int type and in the range [0,6]. The default value is **0**. If the value can be converted into an integer within the range, the time value of the corresponding precision can be output. If the value is invalid, an error is reported.

  Return type: time without time zone

  Implementation method: Register the system function curtime.

  Example:

```
gaussdb=# SELECT curtime(3);
   curtime
--------------
 16:59:57.203
(1 row)
```

  NOTE

  This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- date_add(date,INTERVAL val unit)

Description: Adds a period of time to a specified date and returns the calculation result.

Parameters: For details, see **Table 5 Parameters**.

Table 7-40 Parameters

Parameter	Type	Description	Value Range
date	Expression of the time type, text, datetime, date, or time	Date to be added with an interval.	See the value ranges of the corresponding types.
val	Integer, floating-point number, or character string	Interval to be added.	See the value ranges of the corresponding types.
unit	Keyword	Unit of the interval.	YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE, SECOND and MICROSECOND. For details, see **Time Interval Expressions**.

Return value type: a single return value. For details, see **Table 6 Return value types**.

Table 7-41 Return value types

Return Type	Description
TEXT	The **date** input parameter is of the text type.
DATE	The **date** input parameter is of the date type, and the value of the **unit** input parameter is greater than or equal to **day** (for example, **week** or **month**).
TIME WITHOUT TIMEZONE	The **date** input parameter is of the time type.
DATETIME	The **date** input parameter is of the datetime type, or the **date** input parameter and the value of the **unit** input parameter is less than **day** (for example, **hour** or **second**).

Example:

```
gaussdb=# SELECT DATE_ADD('2018-05-01', INTERVAL 1 DAY);
  date_add
------------
```



```
 2018-05-02
(1 row)
```

NOTE

1. This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.
2. In the prepare statement, the second parameter of **date_add** is an interval expression and cannot be replaced by **$1**, for example, prepare p1 as select date_add('2023-01-01 10:00:00', $1).

- date_add(expr, days)

  Description: Returns a new date by adding a certain number of days to a date.

  Parameters:

  - **expr**: specifies the start date and time. The value type can be expression of the time type, TEXT, DATE, DATETIME, or TIME.
  - **days**: specifies the number of days to be added. The value is of the int type.

  Return type: TEXT, DATE, DATETIME, or TIME WITHOUT TIMEZONE

  Example:

```
gaussdb=# SELECT DATE_ADD('2018-05-01', 1);
  date_add
------------
 2018-05-02
(1 row)
```

  NOTE

  This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- date_format(date, format)

  Description: Outputs the date and time in the specified format.

  Parameters: For details, see **Table 7 Parameters**.

  Table 7-42 Parameters

Parameter	Type	Description	Value Range
date	Expression of the time type, text, datetime, date, or time	Date to be formatted.	See the value ranges of the corresponding types.
format	text	A formatted string.	For details, see **Table 8 Values and meanings of format**.

  - For details about the format parameters, see **Table 7-43**.

Table 7-43 Values and meanings of format

Value	Meaning
%a	Abbreviation of a week (Sun, ..., Sat)
%b	Abbreviation of a month (Jan, ..., Dec)
%c	Month number (0, ..., 12)
%D	Every day in a month with an English prefix (0th, 1st, 2nd, 3rd, ...)
%d	Two-digit representation of every day in a month (00, ..., 31)
%e	Sequence number of every day in a month (0, ..., 31)
%f	Microsecond (000000, ..., 999999)
%H	Hour (00, ..., 23)
%h	Hour (01, ..., 12)
%I	Hour (01, ..., 12)
%i	Minute (00, ..., 59)
%j	Every day in a year (001, ..., 366)
%k	Hours (0, ..., 23)
%l	Hour (1, ..., 12)
%M	Month name (January, ..., December)
%m	Two-digit month (00, ..., 12)
%p	AM or PM
%r	12-hour time (hh:mm:ss followed by AM or PM).
%S	Second (00, ..., 59)
%s	Second (00, ..., 59)
%T	24-hour time (hh:mm:ss)
%U	Week of a year (00, ..., 53). Each week starts from Sunday.
%u	Week of a year (00, ..., 53). Each week starts from Monday.
%V	Week of a year (01, ..., 53). Each week starts from Sunday.
%v	Week of a year (01, ..., 53). Each week starts from Monday.

Value	Meaning
%W	Name of a week (Sunday, ..., Saturday)
%w	Every day in a week (0 = Sunday, ..., 6 = Saturday)
%X	Week of a year. Each week starts from Sunday. The value is a four-digit number and is used for %V.
%x	Week of a year. Each week starts from Monday. The value is a four-digit number and is used for %v.
%Y	Four-digit year
%y	Two-digit year

Return type: text

Example:

```
gaussdb=# SELECT date_format('2023-10-11 12:13:14.151617','%b %c %M %m');
    date_format
--------------------
 Oct 10 October 10
(1 row)
```

NOTE

This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- date_sub(date, INTERVAL val unit)

  Description: Returns a new date by subtracting a certain interval from a date.

  Parameters: For details, see **Table 9 Parameters**.

Table 7-44 Parameters

Parameter	Type	Description	Value Range
date	Expression of the time type, text, datetime, date, or time	Date to be added with an interval.	See the value ranges of the corresponding types.
val	Integer, floating-point number, or character string	Interval to be added.	See the value ranges of the corresponding types.

Parameter	Type	Description	Value Range
unit	Keyword	Unit of the interval.	YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE, SECOND, and MICROSECOND. For details, see **Time Interval Expressions**.

Return type: TEXT, DATE, DATETIME, or TIME WITHOUT TIMEZONE

Example:

```
gaussdb=# SELECT DATE_SUB('2018-05-01', INTERVAL 1 YEAR);
  date_sub
------------
 2017-05-01
(1 row)
```

NOTE

1. This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.
2. In the prepare statement, the second parameter of **date_sub** is an interval expression and cannot be replaced by **$1**, for example, prepare p1 as select date_sub('2023-01-01 10:00:00', $1).

- date_sub(expr, days)

  Description: Specifies the start date and time and the number of days to be subtracted from the start date and time, and returns the subtraction result.

  Parameters:

  - **expr**: specifies the start date and time. The value type can be expression of the time type, TEXT, DATE, DATETIME, or TIME.
  - **days**: specifies the number of days to be subtracted. The value is of the int type.

  Return type: TEXT, DATE, DATETIME, or TIME WITHOUT TIMEZONE

  Example:

  ```
  gaussdb=# SELECT DATE_SUB('2023-1-1', 20);
    date_sub
  ------------
   2022-12-12
  (1 row)
  ```

  NOTE

  This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- datediff(expr1, expr2)

  Description: Returns the number of days between two time expressions.

  Parameters: For details, see **Table 10 Parameters**.

Table 7-45 Parameters

Parameter	Type	Description	Value Range
expr1	Expression of the time type, text, datetime, date, or time	A time expression	See the value ranges of the corresponding types.
expr2	Expression of the time type, text, datetime, date, or time	A time expression	See the value ranges of the corresponding types.

Return type: int

Example:

```
gaussdb=# SELECT datediff('2021-11-12','2021-11-13');
 datediff
----------
       -1
(1 row)
```

NOTE

This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- day()/dayofmonth()

  Description: Extracts the number of days in a date and time expression and returns the result. dayofmonth() is the alias of day() and has the same function.

  Parameter: The input parameter is the date and time to be extracted, which can be an expression of the time type, or of the text, datetime, date, or time type.

  Return type: int

  Example:

```
gaussdb=# SELECT day('2023-01-02');
 day
-----
   2
(1 row)
gaussdb=# SELECT dayofmonth('23-05-22');
 dayofmonth
------------
         22
(1 row)
```

  NOTE

  This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- dayname()

  Description: Returns the name of a day in a week.

Parameter: The input parameter is an expression of the time type, or of the text, datetime, date, or time type.

Return type: text

Example:

```
gaussdb=# SELECT dayname('2023-10-11');
  dayname
-----------
 Wednesday
(1 row)
```

NOTE

- This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.
- The language used by the return value is specified by the GUC parameter **lc_time_names**.

- dayofweek()

  Description: Returns the working day index of a date (1 = Sunday, 2 = Monday, ..., 7 = Saturday).

  Parameter: Expression of the time type, text, datetime, date, or time

  Return type: bigint

  Example:

  ```
  gaussdb=# SELECT dayofweek('2023-04-16');
   dayofweek
  -----------
           1
  (1 row)
  ```

  NOTE

  This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- dayofyear()

  Description: Returns the number of a day in the year.

  Parameter: The input parameter is an expression of the time type, or of the text, datetime, date, or time type.

  Return type: int; value range: 1 to 366.

  Example:

  ```
  gaussdb=# SELECT dayofyear('2000-12-31');
   dayofyear
  -----------
         366
  (1 row)
  ```

  NOTE

  This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- extract(unit FROM date)

  Description: Extracts part of the time. This function can be used in other modes, but the behavior varies greatly. When **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, the function is described as follows:

  Parameters:

- **unit**: text type. For details about the value type, see **Table 7-46**.
- **date**: expression of the time type, such as date, time, datetime, and text.

Return type: bigint

Table 7-46 Values and meanings of unit

Value of UNIT	Description
MICROSECOND	Microsecond
SECOND	Second
MINUTE	Minute
HOUR	Hour
DAY	Day
WEEK	Week
MONTH	Month
QUARTER	Quarter
YEAR	Year
SECOND_MICROSECOND	Concatenated value of second and microsecond
MINUTE_MICROSECOND	Concatenated value of minute, second, and microsecond
MINUTE_SECOND	Concatenated value of minute and second
HOUR_MICROSECOND	Concatenated value of hour, minute, second, and microsecond
HOUR_SECOND	Concatenated value of hour, minute, and second
HOUR_MINUTE	Concatenated value of hour and minute
DAY_MICROSECOND	Concatenated value of day, hour, minute, second, and microsecond
DAY_SECOND	Concatenated value of day, hour, minute, and second
DAY_MINUTE	Concatenated value of day and minute
DAY_HOUR	Concatenated value of day and hour
YEAR_MONTH	Concatenated value of year and month

Value of UNIT	Description
EPOCH	Total number of seconds or interval since 1970-01-01 00:00:00-00 UTC

Example:

```
gaussdb=# SELECT extract(YEAR FROM '2023-10-11');
 extract
---------
    2023
(1 row)

gaussdb=# SELECT extract(QUARTER FROM '2023-10-11');
 extract
---------
       4
(1 row)

gaussdb=# SELECT extract(MONTH FROM '2023-10-11');
 extract
---------
      10
(1 row)

gaussdb=# SELECT extract(WEEK FROM '2023-10-11');
 extract
---------
      41
(1 row)

gaussdb=# SELECT extract(DAY FROM '2023-10-11');
 extract
---------
      11
(1 row)

gaussdb=# SELECT extract(HOUR FROM '2023-10-11 12:13:14');
 extract
---------
      12
(1 row)
```

NOTE

When **sql_compatibility** is set to **'B'**, **b_format_version** is set to **5.7**, and **b_format_dev_version** is set to **s1**, the called function is registered as b_extract. In other cases, the actually registered function is date_part. You can use commands such as **\df b_extract** to query the detailed input parameter and return value of a function.

The GUC parameter **default_week_format** is used to process special week issues. The default value is **0**. For details, see **Table 12 default_week_format**.

Table 7-47 default_week_format

default_week_format	First Day of a Week	Value Range	Which Week is the First Week
0	Sunday	0–53	Week containing Sunday in this year

default_week_format	First Day of a Week	Value Range	Which Week is the First Week
1	Monday	0–53	Week containing four or more days in this year
2	Sunday	1–53	Week containing Sunday in this year
3	Monday	1–53	Week containing four or more days in this year
4	Sunday	0–53	Week containing four or more days in this year
5	Monday	0–53	Week containing Monday in this year
6	Sunday	1–53	Week containing four or more days in this year
7	Monday	1–53	Week containing Monday in this year

- from_days(days)

  Description: Returns the corresponding date value given a number of days.

  Parameter: The input parameter is an expression of the time type, or of the text, datetime, date, or time type.

  Return type: date

  Example:

  ```
  gaussdb=# SELECT from_days(36524);--0099-12-31
   from_days
  ------------
   0099-12-31
  (1 row)
  ```

  NOTE

  - This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.
  - Null characters and zeros are processed as 0. For input parameters that cannot be correctly converted to bigint, an error is reported.
  - If the input parameter is less than **366**, the return date is **0000-00-00**.

- from_unixtime(unix_timestamp[,format])

  Description: Converts a Unix timestamp to the date and time format. A Unix timestamp is the number of seconds from 08:00:00 UTC on January 1, 1970 to a specified time.

  Parameters:

  – **unix_timestamp**: Unix timestamp, which is of the numeric type.

  – **format**: time format. The value is of the text type.

  Return type: text/datetime

Example:

```
gaussdb=# SELECT from_unixtime(1111885200);
    from_unixtime
---------------------
 2005-03-27 09:00:00
(1 row)
```

NOTE

This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- get_format({DATE | TIME | DATETIME | TIMESTAMP}, {'EUR' | 'USA' | 'JIS' | 'ISO' | 'INTERNAL'})

  Description: Converts a date, time, or datetime into a string in a specified time format, that is, the year, month, day, hour, minute, and second formats and sorting standards of different regions.

  Parameters:

  – **DATE|TIME|DATETIME|TIMESTAMP**: time types, which are keywords.
  – **'EUR'|'USA'|'JIS'|'ISO'|'INTERNAL'**: five time formats, which are of the text type.

  Return type: text

  Example:

```
gaussdb=# SELECT get_format(date, 'eur');
 get_format
------------
 %d.%m.%y
(1 row)
gaussdb=# SELECT get_format(date, 'usa');
 get_format
------------
 %m.%d.%y
(1 row)
```

NOTE

1. This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

2. In the prepare statement, the first parameter of **get_format** is a keyword and cannot be replaced by **$1**, for example, prepare p1 as select get_format($1, 'usa').

- hour()

  Description: Returns the hour part of a time after you enter a time type.

  Parameter: Expression of the time type, text, datetime, date, or time

  Return type: bigint

  Example:

```
gaussdb=# SELECT HOUR('10:10:10.1');
 hour
------
   10
(1 row)
```

NOTE

This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- makedate(year,dayofyear)

  Description: Returns a date based on the given year and day.

Parameters:

- **year**: bigint.
- **dayofyear**: bigint, indicating the number of days in the year. The value can cross years. If the value is less than or equal to **0**, **null** is returned.

Return type: date

Example:

```
gaussdb=# SELECT makedate(2000, 60);
  makedate
------------
 2000-02-29
(1 row)
```

NOTE

This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- microsecond()

  Description: Returns the microsecond part of a time after you enter a time type.

  Parameter: Expression of the time type, text, datetime, date, or time

  Return type: bigint

  Example:

```
gaussdb=# SELECT MICROSECOND('2023-5-5 10:10:10.24485');
 microsecond
-------------
      244850
(1 row)
```

NOTE

This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- minute()

  Description: Returns the minute part of a time after you enter a time type.

  Parameter: Expression of the time type, text, datetime, date, or time

  Return type: bigint

  Example:

```
gaussdb=# SELECT MINUTE(time'10:10:10');
 minute
--------
     10
(1 row)
```

NOTE

This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- month()

  Description: Returns the month starting from a past date.

  Parameter: The input parameter is the date and time to be extracted, which can be an expression of the time type, or of the TEXT, DATETIME, DATE, or TIME type.

  Return type: int

Example:

```
gaussdb=# SELECT month('2021-11-30');
 month
-------
    11
(1 row)
```

NOTE

This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- monthname()

  Description: Returns the full month name of a date.

  Parameter: The input parameter is an expression of the time type, or of the text, datetime, date, or time type.

  Return type: text

  Example:

  ```
  gaussdb=# SELECT monthname('2023-02-28');
   monthname
  -----------
   February
  (1 row)
  ```

  NOTE

  - This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.
  - The language used by the return value is controlled by the value of the **lc_time_names** system variable.

- period_add(period, month_number)

  Description: Adds a specified number of months to a specified time segment and returns the result as a time segment.

  Parameters:

  – **period**: bigint, which is a date in the format of YYYYMM or YYMM.

  – **month_number**: bigint, indicating the number of months to be added. The value can be a negative number.

  Return type: bigint, which is a date in YYYYMM format.

  Example:

  ```
  gaussdb=# SELECT period_add(202205, -12);
   period_add
  ------------
      202105
  (1 row)
  ```

  NOTE

  This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- period_diff(p1,p2)

  Description: Calculates the month difference between two time points.

  Parameters: **P1** and **P2** are periods in YYMM or YYYYMM format and are of the bigint type.

  Return type: bigint (month difference)

Example:

```
gaussdb=# SELECT period_diff('202101', '202102');
 period_diff
-------------
          -1
(1 row)
```

NOTE

This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- second()

  Description: Returns the second part of a time after you enter a time type.

  Parameter: Expression of the time type, text, datetime, date, or time

  Return type: bigint

  Example:

```
gaussdb=# SELECT SECOND('2023-5-5 10:10:10');
 second
--------
     10
(1 row)
```

  NOTE

  This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- quarter()

  Description: Returns the quarter part of a date after you enter a date type.

  Parameter: The input parameter is the date and time to be extracted, which can be an expression of the time type, or of the TEXT, DATETIME, DATE, or TIME type.

  Return type: bigint

  Example:

```
gaussdb=# SELECT QUARTER('2012-1-1');
 quarter
---------
       1
(1 row)
```

  NOTE

  This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- str_to_date(str, format)

  Description: Converts a specified character string to a date/time based on the specified date format.

  Parameters:

  – **str**: text type, which is the character string to be formatted into a date.
    format: text type, indicating the target format.
  – The following table lists the format parameters.

Table 7-48 Values and meanings of format

Value	Meaning
%a	Abbreviation of a week (Sun, ..., Sat)
%b	Abbreviation of a month (Jan, ..., Dec)
%c	Month number (0, ..., 12)
%D	Every day in a month with an English prefix (0th, 1st, 2nd, 3rd, ...)
%d	Two-digit representation of every day in a month (00, ..., 31)
%e	Sequence number of every day in a month (0, ..., 31)
%f	Microsecond (000000, ..., 999999)
%H	Hour (00, ..., 23)
%h	Hour (01, ..., 12)
%I	Hour (01, ..., 12)
%i	Minute (00, ..., 59)
%j	Every day in a year (001, ..., 366)
%k	Hours (0, ..., 23)
%l	Hour (1, ..., 12)
%M	Month name (January, ..., December)
%m	Two-digit month (00, ..., 12)
%p	AM or PM
%r	12-hour time (hh:mm:ss followed by AM or PM).
%S	Second (00, ..., 59)
%s	Second (00, ..., 59)
%T	24-hour time (hh:mm:ss)
%U	Week of a year (00, ..., 53). Each week starts from Sunday.
%u	Week of a year (00, ..., 53). Each week starts from Monday.
%V	Week of a year (01, ..., 53). Each week starts from Sunday.
%v	Week of a year (01, ..., 53). Each week starts from Monday.

Value	Meaning
%W	Name of a week (Sunday, ..., Saturday)
%w	Every day in a week (0 = Sunday, ..., 6 = Saturday)
%X	Week of a year. Each week starts from Sunday. The value is a four-digit number and is used for %V.
%x	Week of a year. Each week starts from Monday. The value is a four-digit number and is used for %v.
%Y	Four-digit year
%y	Two-digit year

Return type: text

Example:

```
gaussdb=# SELECT str_to_date('May 1, 2013','%M %d,%Y');--2013-05-01
 str_to_date
-------------
 2013-05-01
(1 row)
```

NOTE

- This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.
- Only the time in YYYY-MM-DD format can be returned.
- If the time contains 0 year, 0 month, and 0 day or contains only hour, minute, and second, an alarm is generated and **NULL** is returned.

- subdate(expr, days)

  Description: Specifies the start date and time and the number of days to be subtracted from the start date and time, and returns the subtraction result.

  Parameters:

  – **expr**: specifies the start date and time. The value type can be expression of the time type, TEXT, DATE, DATETIME, or TIME.

  – **days**: specifies the number of days to be subtracted. The value is of the int type.

  Return type: TEXT, DATE, DATETIME, or TIME WITHOUT TIMEZONE

  Example:

  ```
  gaussdb=# SELECT SUBDATE('2023-1-1', 20);
    subdate
  ------------
   2022-12-12
  (1 row)
  ```

  NOTE

  This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- subdate(expr,INTERVAL val unit)

  Description: Specifies the start date and time and the interval to be subtracted from the start date and time, and returns the subtraction result.

  Parameters: For details, see **Table 7-49**.

  Table 7-49 Parameters

Parameter	Type	Description	Value Range
expr	Expression of the time type, text, datetime, date, or time	Specifies the start date and time.	See the value ranges of the corresponding types.
val	Integer, floating-point number, or character string	Specifies the interval to be subtracted.	See the value ranges of the corresponding types.
unit	Keyword		YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE, SECOND, and MICROSECOND. For details, see **Time Interval Expressions**.

  Return type: TEXT, DATE, DATETIME, or TIME WITHOUT TIMEZONE

  Example:

```
gaussdb=# SELECT SUBDATE('2018-05-01', INTERVAL 1 YEAR);
  subdate
------------
 2017-05-01
(1 row)
```

  NOTE

  1. This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

  2. In the prepare statement, the second parameter of **subdate** is an interval expression. When parameter **$1** is used to completely replace **subdate**, the result is unexpected, for example, prepare p1 as select subdate('2023-01-01 10:00:00', $1); execute p1(interval 13 hour). The unexpected result returned in this test case is **'2023-01-01 10:00:00'**.

- subtime(expr1,expr2)

  Description: Returns the difference between **expr1** and **expr2**.

  Parameters:

  – **expr1** is an expression of the timestamp without time zone or time type, and **expr2** is a time expression.

  – The return value type is related to the type of **expr1**. If the two input parameters are of the text type, the return value type is text. If the two input parameters are parsed as timestamp without time zone, the return

value type is timestamp without time zone. If the two input parameters are parsed as time, the return value type is time.

Return type: text, timestamp without time zone, or time.

Example:

```
gaussdb=# SELECT subtime('2000-03-01 20:59:59', '22:58');
      subtime
---------------------
 2000-02-29 22:01:59
(1 row)
```

NOTE

This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- addtime(expr1,expr2)

  Description: Returns the sum of **expr1** and **expr2**. The format of the return value is the same as that of **expr1**.

  Parameters:

  - **expr1** is an expression of the timestamp without time zone or time type, and **expr2** is a time expression.
  - The return value type is related to the type of **expr1**. If it is parsed as timestamp without time zone, the return value type is timestamp without time zone. If it is parsed as time, the return value type is time.

  Return type: text, timestamp without time zone, or time.

  Example:

  ```
  gaussdb=# SELECT addtime('2000-03-01 20:59:59', '00:00:01');
        addtime
  ---------------------
   2000-03-01 21:00:00
  (1 row)
  ```

  NOTE

  This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- time_format(time, format)

  Description: Formats the **time** input parameter based on the format specified by **format**.

  Parameters:

  - The value type of **time** is expression of the time type, text, datetime, date, or time
  - **format** is of the text type. **Table 7-50** lists the supported formats.

  Table 7-50 Values and meanings of format

Value	Meaning
%f	Microsecond (000000–999999)
%H	Hour (00 to 23)
%h, %I	Hour (00 to 12)

Value	Meaning
%l	Hour (0 to 12)
%k	Hour (0 to 838)
%i	Minute (00 to 59)
%p	AM or PM
%r	Time in 12-hour AM or PM format (hh:mm:ss AM/PM)
%S, %s	Second (00 to 59)
%T	Time in 24-hour format (hh:mm:ss)
%a, %b, %D, %j, %M, %U, %u, %V, %v, %W, %w, %X, %x	NULL.
%c, %e	0
%d, %m, %y	00
%Y	0000
%*Other characters*|*Other characters*, for example, %A/A	The character itself is returned, for example, A.
%*Single character* + string s	%*Single character* is parsed and then concatenated with s.

Return type: text

Example:

```
gaussdb=# SELECT TIME_FORMAT('25:30:30', '%T|%r|%H|%h|%I|%i|%S|%f|%p|%k');
                time_format
-----------------------------------------------------
 25:30:30|01:30:30 AM|25|01|01|30|30|000000|AM|25
(1 row)
```

NOTE

This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- time_to_sec()

  Description: Converts the input parameter of the time type to the number of seconds.

  Parameter: The input parameter is an expression of the time type, or of the text, datetime, date, or time type. The time expression is parsed as time.

  Return type: int

  Example:

```
gaussdb=# SELECT time_to_sec('00:00:01');
 time_to_sec
-------------
           1
(1 row)
```


> **NOTE**
>
> This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- timediff()

  Description: Calculates the difference between two time points and returns an interval.

  Parameter: There are two parameters, which are of the time expression, text, datetime, date, or time type.

  Return type: TIME WITHOUT TIMEZONE

  Example:

```
gaussdb=# SELECT timediff(date'2022-12-30',20221229);
 timediff
----------
 24:00:00
(1 row)
```

> **NOTE**
>
> This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- timestampadd(unit,interval,datetime_expr)

  Description: Returns a new timestamp calculated by adding multiple intervals of a unit to **datetime_expr**.

  Parameters: For details, see **Table 16 Parameters**.

Table 7-51 Parameters

Parameter	Type	Description	Value Range
unit	Keyword	Unit of the interval.	YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE, SECOND, MICROSECOND, Y, MM (month), D, H, M (minute), S, US, YRS, QTR, MON, HRS, MIN, YEARS, WEEKS, or HOURS.
interval	numeric	Interval.	See the value ranges of the corresponding types.

Parameter	Type	Description	Value Range
datetime_expr	Expression of the time type, text, datetime, date, or time	Date and time to be changed. If the value is of the text type, the return value type is text. If the value is of the time type, the return value type is time. In other cases, the return value type is datetime.	See the value ranges of the corresponding types.

Return type: DATETIME, TEXT, TIME WITHOUT TIMEZONE

Example:

```
gaussdb=# SELECT TIMESTAMPADD(DAY,-2,'2022-07-27');
 timestampadd
--------------
 2022-07-25
(1 row)
```

NOTE

- This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.
- The actually registered function of timestampadd is timestamp_add. You can run commands such as **\df timestamp_add** to query the detailed input parameter and return value of the function.
- In the prepare statement, the first parameter of **timestampadd** is a keyword and cannot be replaced by **$1**, for example, prepare p1 as select timestampadd($1, -2, '2023-01-01');.

- to_days()

  Description: Returns the number of days from year 0 of a specified date.

  Parameter: The input parameter is an expression of the time type, or of the text, datetime, date, or time type. The time expression is parsed as date.

  Return type: bigint

  Example:

```
gaussdb=# SELECT to_days('2000-1-1');
 to_days
---------
  730485
(1 row)
```


> **NOTE**
>
> This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- to_seconds()

  Description: Returns the number of seconds since the year 0 A.D.

  Parameter: The input parameter is an expression of the time type, or of the text, datetime, date, or time type. The time expression is parsed as datetime.

  Return type: bigint

  Example:

```
gaussdb=# SELECT TO_SECONDS('2009-11-29 13:43:32');
 to_seconds
-------------
 63426721412
(1 row)
```

  > **NOTE**
  >
  > This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- unix_timestamp([date])

  Description: Returns a Unix timestamp representing the number of seconds since "1970-01-01 08:00" UTC. If there is no input parameter, the default value is the datetime timestamp when the function is called.

  Parameter: Expression of the time type, text, datetime, date, or time

  Return type: numeric

  Example:

```
gaussdb=# SELECT UNIX_TIMESTAMP('2022-12-22');
 unix_timestamp
----------------
     1671638400
(1 row)
```

  > **NOTE**
  >
  > This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- utc_date()

  Description: Returns the current UTC date of function execution as a value in YYYY-MM-DD format.

  Return type: date

  Example:

```
gaussdb=# SELECT utc_date();
  utc_date
------------
 2023-08-10
(1 row)
```

  > **NOTE**
  >
  > This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- utc_time()

Description: Returns the current UTC time of function execution as a value in HH:MM:SS format.

Parameter: Indicates the time precision. The value is of the int type and in the range [0,6]. The default value is **0**.

Return type: time without time zone

Example:

```
gaussdb=# SELECT utc_time();
 utc_time
----------
 11:47:53
(1 row)
```

NOTE

This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- utc_timestamp()

  Description: Returns the current UTC timestamp of function execution as a value in YYYY-MM-DD HH:MM:SS format.

  Parameter: Indicates the time precision. The value is of the int type and in the range [0,6]. The default value is **0**.

  Return type: timestamp without time zone

  Example:

```
gaussdb=# SELECT utc_timestamp();
    utc_timestamp
---------------------
 2023-08-21 11:51:19
(1 row)
```

  NOTE

  This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- week(date[, mode])

  Description: Returns the number of weeks of a date.

  Parameters:

  – **date**: specifies the date and time, which is of the time expression, text, datetime, date, or time type.

  – Table 7-52 describes the optional parameter **mode**. The default value is **0**.

  Table 7-52 Description of mode

mode	First Day of a Week	Value Range	Which Week is the First Week
0	Sunday	0–53	Week containing Sunday in this year
1	Monday	0–53	Week containing four or more days in this year

mode	First Day of a Week	Value Range	Which Week is the First Week
2	Sunday	1–53	Week containing Sunday in this year
3	Monday	1–53	Week containing four or more days in this year
4	Sunday	0–53	Week containing four or more days in this year
5	Monday	0–53	Week containing Monday in this year
6	Sunday	1–53	Week containing four or more days in this year
7	Monday	1–53	Week containing Monday in this year

Return type: bigint

Example:

```
gaussdb=# SELECT week(date'2000-01-01', 1);
 week
------
    0
(1 row)

gaussdb=# SELECT week('2000-01-01', 2);
 week
------
   52
(1 row)
```

NOTE

This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- weekday()

  Description: Returns the working day index of a date, that is, Monday is 0, Tuesday is 1, Wednesday is 2, Thursday is 3, Friday is 4, Saturday is 5, and Sunday is 6.

  Parameter: Expression of the time type, text, datetime, date, or time

  Return type: bigint

  Example:

```
gaussdb=# SELECT weekday('1970-01-01 12:00:00');
 weekday
---------
       3
(1 row)
```


> **NOTE**
>
> This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- weekofyear(date)

  Description: Returns the calendar week of the date and time. The value ranges from 1 to 53. It is equivalent to week(date, 3).

  Parameters:

  - **date**: specifies the date and time, which is of the time expression, text, datetime, date, or time type.
  - This function is equivalent to week(date, 3). For details, see **week(date[, mode])**.

  Return type: bigint

  Example:

```
gaussdb=# SELECT weekofyear('1970-05-22');
 weekofyear
------------
         21
(1 row)
```

  > **NOTE**
  >
  > This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- year()

  Description: Extracts the year part of the date and time and returns the result.

  Parameter: The input parameter is the date and time to be extracted, which can be an expression of the time type, or of the text, datetime, date, or time type.

  Return type: int

  Example:

```
gaussdb=# SELECT year('23-05-22');
 year
------
 2023
(1 row)
```

  > **NOTE**
  >
  > This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- yearweek(date[, mode])

  Description: Returns the year and week of a date.

  Parameters:

  - **date**: specifies the date and time.
  - The value of **date** can be DATE, DATETIME, TIMESTAMP, TIME, TEXT or NUMERIC.
  - **Table 7-53** describes the optional parameter **mode**. The default value is **0**.

Table 7-53 Description of mode

mode	First Day of a Week	Value Range	Which Week is the First Week
0	Sunday	0–53	Week containing Sunday in this year
1	Monday	0–53	Week containing four or more days in this year
2	Sunday	1–53	Week containing Sunday in this year
3	Monday	1–53	Week containing four or more days in this year
4	Sunday	0–53	Week containing four or more days in this year
5	Monday	0–53	Week containing Monday in this year
6	Sunday	1–53	Week containing four or more days in this year
7	Monday	1–53	Week containing Monday in this year

Return type: bigint

Example:

```
gaussdb=# SELECT yearweek(datetime'2000-01-01', 3);
 yearweek
----------
   199952
(1 row)
```

NOTE

This function is valid only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

Table 7-54 shows the templates for truncating date/time values.

Table 7-54 Truncating date/time values

Item	Format	Description
Microsecond	MICROSECON	Truncates date/time values, accurate to the microsecond (000000–999999).
	US	
	USEC	

Item	Format	Description
	USECOND	
Millisecond	MILLISECON	Truncates date/time values, accurate to the millisecond (000–999).
	MS	
	MSEC	
	MSECOND	
Second	S	Truncates date/time values, accurate to the second (00–59).
	SEC	
	SECOND	
Minute	M	Truncates date/time values, accurate to the minute (00–59).
	MI	
	MIN	
	MINUTE	
Hour	H	Truncates date/time values, accurate to the hour (00–23).
	HH	
	HOUR	
	HR	
Day	D	Truncates date/time values, accurate to the day (01-01 to 12–31)
	DAY	
	DD	
	DDD	
	J	
Week	W	Truncates date/time values, accurate to the week (the first day of the current week).
	WEEK	
Month	MM	Truncates date/time values, accurate to the month (the first day of the current month).
	MON	
	MONTH	
Quarter	Q	Truncates date/time values, accurate to the quarter (the first day of the current quarter).
	QTR	
	QUARTER	

Item	Format	Description
Year	Y	Truncates date/time values, accurate to the year (the first day of the current year).
	YEAR	
	YR	
	YYYY	
Decade	DEC	Truncates date/time values, accurate to the decade (the first day of the current decade).
	DECADE	
Century	C	Truncates date/time values, accurate to the century (the first day of the current century).
	CC	
	CENT	
	CENTURY	
Millennium	MIL	Truncates date/time values, accurate to the millennium (the first day of the current millennium).
	MILLENNIA	
	MILLENNIUM	

Table 7-55 Parameters for time truncation and rounding

Item	Format	Description
Minute	M	Truncated or rounded off, accurate to minute (00–59).
	MI	
	MIN	
	MINUTE	
Hour	H	Truncated or rounded off, accurate to hour (00–23).
	HH	
	HOUR	
	HR	
	HH12	
	HH24	
Day	DD	Truncated or rounded off, accurate to day (01-01 to 12-31).
	DDD	
	J	

Item	Format	Description
ISO week	IW	Truncated or rounded off, accurate to week (the first day of the week is Monday).
Week	DAY	Truncated or rounded off, accurate to week (the first day of the week is Sunday).
	DY	
	D	
Week of the month	W	Truncated or rounded off, accurate to week (the first day of the week is the first day of the month).
Week of the year	WW	Truncated or rounded off, accurate to week (the first day of the week is the first day of the year).
Month	MM	Truncated or rounded off, accurate to month (the first day of the month).
	MON	
	MONTH	
	RM	
Quarter	Q	Truncated or rounded off, accurate to quarter (the first day of the quarter).
	QTR	
	QUARTER	
Year	Y	Truncated or rounded off, accurate to year (the first day of the current year).
	YEAR	
	YR	
	YYYY	
	SYYYY	
	YYY	
	YY	
	SYEAR	
Decade	DEC	Truncated or rounded off, accurate to decade (the first day of the current decade).
	DECADE	
Century	C	Truncated or rounded off, accurate to the century (the first day of the century).
	CC	
	CENT	
	CENTURY	

Item	Format	Description
	SCC	
Millennium	MIL	Truncated or rounded off, accurate to millennium (the first day of the millennium).
	MILLENNIA	
	MILLENNIUM	

NOTE

The behaviors of **Table 7-55** are valid only when the value of **a_format_version** is **10c** and that of **a_format_dev_version** is **s1** in an A-compatible database.

- timestamp_diff(text, timestamp, timestamp)

  Description: Calculates the difference between two timestamps and truncates the difference to the precision specified by text.

  Return type: bigint

  Example:

```
gaussdb=# SELECT timestamp_diff('year','2018-01-01','2020-04-01');
 timestamp_diff
----------------
              2
(1 row)
gaussdb=# SELECT timestamp_diff('month','2018-01-01','2020-04-01');
 timestamp_diff
----------------
             27
(1 row)
gaussdb=# SELECT timestamp_diff('quarter','2018-01-01','2020-04-01');
 timestamp_diff
----------------
              9
(1 row)
gaussdb=# SELECT timestamp_diff('week','2018-01-01','2020-04-01');
 timestamp_diff
----------------
            117
(1 row)
gaussdb=# SELECT timestamp_diff('day','2018-01-01','2020-04-01');
 timestamp_diff
----------------
            821
(1 row)
gaussdb=# SELECT timestamp_diff('hour','2018-01-01 10:10:10','2018-01-01 12:12:12');
 timestamp_diff
----------------
              2
(1 row)
gaussdb=# SELECT timestamp_diff('minute','2018-01-01 10:10:10','2018-01-01 12:12:12');
 timestamp_diff
----------------
            122
(1 row)
gaussdb=# SELECT timestamp_diff('second','2018-01-01 10:10:10','2018-01-01 10:12:12');
 timestamp_diff
----------------
            122
(1 row)
gaussdb=# SELECT timestamp_diff('microsecond','2018-01-01 10:10:10','2018-01-01 10:12:12');
 timestamp_diff
```



```
----------------
    122000000
(1 row)
```

TIMESTAMPDIFF

- **TIMESTAMPDIFF(***unit , timestamp_expr1, timestamp_expr2***)**

Description: Returns the result of **timestamp_expr2** – **timestamp_expr1** in the specified unit. This function is equivalent to **timestamp_diff(text, timestamp, timestamp)**.

Parameters: **timestamp_expr1** and **timestamp_expr2** are of the time expression, text, datetime, date, or time type. **unit** specifies the unit of the difference between two dates.

Return type: bigint

NOTE

- This function takes effect only in databases in B compatibility mode.
- When **sql_compatibility** is set to **B**, **b_format_version** is set to **5.7**, and **b_format_dev_version** is set to **s1**, the called function is registered as b_timestampdiff. If the GUC parameter is not enabled in a database in B compatibility mode, the called function is registered as timestamp_diff. You can run the **\df b_timestampdiff** command to query the detailed input parameter and return value of the function.

- year

  Year.

  ```
  gaussdb=#  SELECT TIMESTAMPDIFF(YEAR, '2018-01-01', '2020-01-01');
   timestamp_diff
  ----------------
               2
  (1 row)
  ```

- quarter

  Quarter.

  ```
  gaussdb=#  SELECT TIMESTAMPDIFF(QUARTER, '2018-01-01', '2020-01-01');
   timestamp_diff
  ----------------
               8
  (1 row)
  ```

- month

  Month.

  ```
  gaussdb=#  SELECT TIMESTAMPDIFF(MONTH, '2018-01-01', '2020-01-01');
   timestamp_diff
  ----------------
              24
  (1 row)
  ```

- week

  Week.

  ```
  gaussdb=#  SELECT TIMESTAMPDIFF(WEEK, '2018-01-01', '2020-01-01');
   timestamp_diff
  ----------------
             104
  (1 row)
  ```

- day

  Day.

```
gaussdb=#  SELECT TIMESTAMPDIFF(DAY, '2018-01-01', '2020-01-01');
 timestamp_diff
----------------
            730
(1 row)
```

- hour

  Hour.

```
gaussdb=#  SELECT TIMESTAMPDIFF(HOUR, '2020-01-01 10:10:10', '2020-01-01 11:11:11');
 timestamp_diff
----------------
              1
(1 row)
```

- minute

  Minute.

```
gaussdb=#  SELECT TIMESTAMPDIFF(MINUTE, '2020-01-01 10:10:10', '2020-01-01 11:11:11');
 timestamp_diff
----------------
             61
(1 row)
```

- second

  Second.

```
gaussdb=#  SELECT TIMESTAMPDIFF(SECOND, '2020-01-01 10:10:10', '2020-01-01 11:11:11');
 timestamp_diff
----------------
           3661
(1 row)
```

- microseconds

  The second field, including fractional parts, is multiplied by 1,000,000.

```
gaussdb=#  SELECT TIMESTAMPDIFF(MICROSECOND, '2020-01-01 10:10:10.000000', '2020-01-01
10:10:10.111111');
 timestamp_diff
----------------
         111111
(1 row)
```

EXTRACT

- **EXTRACT(***field* **FROM** *source***)**

The **extract** function retrieves fields such as year or hour from date/time values. **source** must be a value expression of the date type (**timestamp**, **time**, or **interval**). **field** is an identifier or string that selects what field to extract from the source value. The extract function returns a value of the double precision type. The following are valid **field** names:

- century

  Century.

  The first century starts at 0001-01-01 00:00:00 AD. This definition applies to all Gregorian calendar countries. There is no century number 0. You go from **-1** century to **1** century.

```
gaussdb=# SELECT EXTRACT(CENTURY FROM TIMESTAMP '2000-12-16 12:21:13');
 date_part
-----------
        20
(1 row)
```


- day
  - For **timestamp** values, the day (of the month) field (1–31)

    ```
    gaussdb=# SELECT EXTRACT(DAY FROM TIMESTAMP '2001-02-16 20:38:40');
     date_part
    -----------
            16
    (1 row)
    ```

  - For **interval** values, the number of days

    ```
    gaussdb=# SELECT EXTRACT(DAY FROM INTERVAL '40 days 1 minute');
     date_part
    -----------
            40
    (1 row)
    ```

- decade

  Year divided by 10

  ```
  gaussdb=# SELECT EXTRACT(DECADE FROM TIMESTAMP '2001-02-16 20:38:40');
   date_part
  -----------
         200
  (1 row)
  ```

- dow

  Day of the week as Sunday (**0**) to Saturday (**6**)

  ```
  gaussdb=# SELECT EXTRACT(DOW FROM TIMESTAMP '2001-02-16 20:38:40');
   date_part
  -----------
           5
  (1 row)
  ```

- doy

  Day of the year (1–365 or 366)

  ```
  gaussdb=# SELECT EXTRACT(DOY FROM TIMESTAMP '2001-02-16 20:38:40');
   date_part
  -----------
          47
  (1 row)
  ```

- epoch
  - For **timestamp with time zone** values, the number of seconds since 1970-01-01 00:00:00-00 UTC (can be negative).

    For **date** and **timestamp** values, the number of seconds since 1970-01-01 00:00:00-00 local time.

    For **interval** values, the total number of seconds in the interval.

    ```
    gaussdb=# SELECT EXTRACT(EPOCH FROM TIMESTAMP WITH TIME ZONE '2001-02-16
    20:38:40.12-08');
      date_part
    --------------
     982384720.12
    (1 row)
    gaussdb=# SELECT EXTRACT(EPOCH FROM INTERVAL '5 days 3 hours');
     date_part
    -----------
        442800
    (1 row)
    ```

  - Way to convert an epoch value back to a timestamp

    ```
    gaussdb=# SELECT TIMESTAMP WITH TIME ZONE 'epoch' + 982384720.12 * INTERVAL '1
    second' AS RESULT;
              result
    ---------------------------
     2001-02-17 12:38:40.12+08
    (1 row)
    ```


- hour

  Hour (0–23)

```
gaussdb=# SELECT EXTRACT(HOUR FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
        20
(1 row)
```

- isodow

  Day of the week (1–7)

  Monday is 1 and Sunday is 7.

  NOTE

  This is identical to **dow** except for Sunday.

```
gaussdb=# SELECT EXTRACT(ISODOW FROM TIMESTAMP '2001-02-18 20:38:40');
 date_part
-----------
         7
(1 row)
```

- isoyear

  The ISO 8601 year that the date falls in (not applicable to intervals).

  Each ISO year begins with the Monday of the week containing January 4, so in early January or late December the ISO year may be different from the Gregorian year. See **week** for more information.

```
gaussdb=# SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-01');
 date_part
-----------
      2005
(1 row)
gaussdb=# SELECT EXTRACT(WEEK FROM TIMESTAMP '2006-01-01 00:00:40');
 date_part
-----------
        52
(1 row)
gaussdb=# SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-02');
 date_part
-----------
      2006
(1 row)
gaussdb=# SELECT EXTRACT(WEEK FROM TIMESTAMP '2006-01-02 00:00:40');
 date_part
-----------
         1
(1 row)
```

- microseconds

  The second field, including fractional parts, is multiplied by 1,000,000.

```
gaussdb=# SELECT EXTRACT(MICROSECONDS FROM TIME '17:12:28.5');
 date_part
-----------
  28500000
(1 row)
```

- millennium

  Millennium.

  Years in the 1900s are in the second millennium. The third millennium started from January 1, 2001.

```
gaussdb=# SELECT EXTRACT(MILLENNIUM FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
```



```
       3
(1 row)
```

- milliseconds

  Second field, including fractional parts, is multiplied by 1000. Note that this includes full seconds.

  ```
  gaussdb=# SELECT EXTRACT(MILLISECONDS FROM TIME '17:12:28.5');
   date_part
  -----------
      28500
  (1 row)
  ```

- minute

  Minute (0–59).

  ```
  gaussdb=# SELECT EXTRACT(MINUTE FROM TIMESTAMP '2001-02-16 20:38:40');
   date_part
  -----------
         38
  (1 row)
  ```

- month

  For **timestamp** values, the specific month in the year (1–12).

  ```
  gaussdb=# SELECT EXTRACT(MONTH FROM TIMESTAMP '2001-02-16 20:38:40');
   date_part
  -----------
          2
  (1 row)
  ```

  For **interval** values, the number of months, modulo 12 (0–11).

  ```
  gaussdb=# SELECT EXTRACT(MONTH FROM INTERVAL '2 years 13 months');
   date_part
  -----------
          1
  (1 row)
  ```

- quarter

  Quarter of the year (1–4) that the date is in.

  ```
  gaussdb=# SELECT EXTRACT(QUARTER FROM TIMESTAMP '2001-02-16 20:38:40');
   date_part
  -----------
          1
  (1 row)
  ```

- second

  Second field, including fractional parts (0–59).

  ```
  gaussdb=# SELECT EXTRACT(SECOND FROM TIME '17:12:28.5');
   date_part
  -----------
       28.5
  (1 row)
  ```

- timezone

  Time zone offset from UTC, measured in seconds. Positive values correspond to time zones east of UTC, negative values to zones west of UTC.

- timezone_hour

  Hour component of the time zone offset.

- timezone_minute

  Minute component of the time zone offset.

- week

Number of the week of the year that the day is in. By definition (ISO 8601), the first week of a year contains January 4 of that year. In other words, the first Thursday of a year is in week 1 of that year.

Because of this, it is possible for early January dates to be part of the 52nd or 53rd week of the previous year, and late December dates to be part of the 1st week of the next year. For example, 2006-01-01 is the 52nd week of 2005, and 2006-01-02 is the first week of 2006. You are advised to use the columns **isoyear** and **week** together to ensure consistency.

```
gaussdb=# SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-01');
 date_part
-----------
      2005
(1 row)
gaussdb=# SELECT EXTRACT(WEEK FROM TIMESTAMP '2006-01-01 00:00:40');
 date_part
-----------
        52
(1 row)
gaussdb=# SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-02');
 date_part
-----------
      2006
(1 row)
gaussdb=# SELECT EXTRACT(WEEK FROM TIMESTAMP '2006-01-02 00:00:40');
 date_part
-----------
         1
(1 row)
```

- year

  Year field.

```
gaussdb=# SELECT EXTRACT(YEAR FROM TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
      2001
(1 row)
```

date_part

The date_part function is modeled on the traditional Ingres equivalent to the SQL-standard function extract:

- **date_part('***field***,** *source***)**

Note that here the **field** parameter needs to be a string value, not a name. The valid field names are the same as those for **extract**. For details, see **EXTRACT**.

Example:

```
gaussdb=# SELECT date_part('day', TIMESTAMP '2001-02-16 20:38:40');
 date_part
-----------
        16
(1 row)
gaussdb=# SELECT date_part('hour', INTERVAL '4 hours 3 minutes');
 date_part
-----------
         4
(1 row)
```

Table 7-56 specifies the formats for formatting date and time values.

Table 7-56 Formats for formatting date and time

Category	Format	Description
Hour	HH	Number of hours in one day (01–12)
	HH12	Number of hours in one day (01–12)
	HH24	Number of hours in one day (00–23)
Minute	MI	Minute (00–59)
Second	SS	Second (00–59)
	FF	Microsecond (000000–999999)
	FF1	Microsecond (0-9)
	FF2	Microsecond (00-99)
	FF3	Microsecond (000-999)
	FF4	Microsecond (0000-9999)
	FF5	Microsecond (00000-99999)
	FF6	Microsecond (000000–999999)
	SSSSS	Second after midnight (0–86399)
Morning and afternoon	AM or A.M.	Morning identifier
	PM or P.M.	Afternoon identifier
Year	Y,YYY	Year with comma (with four digits or more)
	SYYYY	Year with four digits BC
	YYYY	Year (with four digits or more)
	YYY	Last three digits of a year
	YY	Last two digits of a year
	Y	Last one digit of a year
	IYYY	ISO year (with four digits or more)
	IYY	Last three digits of an ISO year
	IY	Last two digits of an ISO year
	I	Last one digit of an ISO year
	RR	Last two digits of a year (A year of the 20th century can be stored in the 21st century.)
	RRRR	Capable of receiving a year with four digits or two digits. If there are 2 digits, the value is the same as the returned value of RR. If there are 4 digits, the value is the same as YYYY.

Category	Format	Description
	• BC or B.C. • AD or A.D.	Era indicator BC or AD
Month	MONTH	Full name of a month in uppercase (9 characters are filled in if the value is empty.)
	MON	Month in abbreviated format in uppercase (with three characters)
	MM	Month (01–12)
	RM	Month in Roman numerals (I–XII; I=JAN) and uppercase
Day	DAY	Full name of a date in uppercase (9 characters are filled in if the value is empty.)
	DY	Day in abbreviated format in uppercase (with three characters)
	DDD	Day in a year (001–366)
	DD	Day in a month (01–31)
	D	Day in a week (1–7).
Week	W	Week in a month (1–5) (The first week starts from the first day of the month.)
	WW	Week in a year (1–53) (The first week starts from the first day of the year.)
	IW	Week in an ISO year (The first Thursday is in the first week.)
Century	CC	Century (with two digits) (The 21st century starts from 2001-01-01.)
Julian date	J	Julian date (starting from January 1 of 4712 BC)
Quarter	Q	Quarter

When **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1** in an A-compatible database, the date and time will be formatted in the specified pattern.

Table 7-57 New formats for formatting date and time

Item	Format	Description
Century	SCC	Century. A hyphen (-) will be displayed before BC years.

Item	Format	Description
Year	SYYYY	Returns a numeric year. A hyphen (-) will be displayed before BC years.
	RR	Returns the two-digit year of a date.
	RRRR	Returns the four-digit year of a date.
	YEAR	Returns the year of the character type.
	SYEAR	Returns the year of the character type. A hyphen (-) will be displayed before BC years.
Date Format	DL	Returns the date in the specified long date format.
	DS	Returns the date in the specified short date format.
	TS	Returns the time in the specified time format.
Second	FF7	Microsecond (0000000-9999990)
	FF8	Microsecond (00000000-99999900)
	FF9	Microsecond (000000000-999999000)

NOTE

In the table, the rules for RR to calculate years are as follows:

- If the range of the input two-digit year is between 00 and 49:

  If the last two digits of the current year are between 00 and 49, the first two digits of the returned year are the same as the first two digits of the current year.

  If the last two digits of the current year are between 50 and 99, the first two digits of the returned year equal to the first two digits of the current year plus 1.

- If the range of the input two-digit year is between 50 and 99:

  If the last two digits of the current year are between 00 and 49, the first two digits of the returned year equal to the first two digits of the current year minus 1.

  If the last two digits of the current year are between 50 and 99, the first two digits of the returned year are the same as the first two digits of the current year.

NOTE

In the scenario where this function is in an A-compatible database, the value of **a_format_version** is **10c**, and that of **a_format_dev_version** is **s1**:

- The **to_date** and **to_timestamp** functions support the FX pattern (the input strictly corresponds to a pattern) and the X pattern (decimal point).
- The input pattern can appear only once, indicating that the patterns of the same information cannot appear at the same time. For example, SYYYY and BC cannot be used together.
- The pattern is case insensitive.
- You are advised to use a separator between the input and the pattern. Otherwise, the behavior may not be compatible with database O.

7.6.9 Type Conversion Functions

Type Conversion Functions

- cash_words(money)

  Description: Type conversion function, which converts money into text.

  Example:

```
gaussdb=# SELECT cash_words('1.23');
            cash_words
-----------------------------------
 One dollar and twenty three cents
(1 row)
```

- convert(expr, type)

  Description: Converts **expr** to the type specified by **type**.

  Parameter: The first parameter is an arbitrary value, and the second parameter is the type name.

  Return value type: The return value type is the same as that of the input.

  Example:

```
gaussdb=# SELECT convert(12.5, text);
 text
------
 12.5
(1 row)
```

  NOTE

  This function takes effect only in databases in B compatibility mode.

- cast(x as y [DEFAULT z ON CONVERSION ERROR][,fmt])

  Description: Converts *x* into the type specified by *y*. When **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, if **y** is of the char type, this function converts **x** to the varchar type.

  DEFAULT z ON CONVERSION ERROR: This parameter is optional. If the attempt to convert *x* to type *y* fails, *z* is converted to type *y* by default.

  fmt: This parameter is optional, which can be specified when *y* is one of the following data types:

  int1/int2/int4/int8/int16/float4/float8/numeric: The function of the optional parameter *fmt* is the same as that in the to_number(expr [,*fmt*]) function.

date/timestamp/timestamp with time zone: The function of the optional parameter *fmt* is the same as that in the to_date(string [,*fmt*])/ to_timestamp(string [,*fmt*]) /to_timestamp_tz(string [,*fmt*]) function.

Example:

```
gaussdb=# SELECT cast('22-oct-1997' as timestamp);
      timestamp
---------------------
 1997-10-22 00:00:00
(1 row)

gaussdb=# SELECT cast('22-ocX-1997' as timestamp DEFAULT '22-oct-1997' ON CONVERSION
ERROR, 'DD-Mon-YYYY');
      timestamp
---------------------
 1997-10-22 00:00:00
(1 row)

gaussdb=# CREATE DATABASE gaussdb_m  WITH dbcompatibility  'b';
gaussdb=# \c gaussdb_m
-- Set compatible version control parameters.
gaussdb_m=# SET b_format_version='5.7';
gaussdb_m=# SET b_format_dev_version='s1';
gaussdb_m=# SELECT cast('aaa' as char);
 varchar
---------
 aaa
(1 row)
```

NOTE

The DEFAULT z ON CONVERSION ERROR and *fmt* syntaxes are supported only when **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1**.

- cast(x AS {SIGNED | UNSIGNED} [INT | INTEGER])

  Description: Converts **x** to the BIGINT SIGNED or BIGINT UNSIGNED type.

  Return type: BIGINT SIGNED or BIGINT UNSIGNED

  Example:

  ```
  gaussdb=# SELECT CAST(12 AS UNSIGNED);
   uint8
  -------
   12
  (1 row)
  ```

- hextoraw(text)

  Description: Converts a string in hexadecimal format into raw type.

  Return type: raw

  Example:

  ```
  gaussdb=# SELECT hextoraw('7D');
   hextoraw
  ----------
   7D
  (1 row)
  ```

- numtoday(numeric)

  Description: Converts values of the number type into the timestamp of the specified type.

  Return type: timestamp

  Example:

  ```
  gaussdb=# SELECT numtoday(2);
   numtoday
  ```



```
----------
 2 days
(1 row)
```

- rawtohex(string)

  Description: Converts a string in binary format into hexadecimal format.

  The result is the ACSII code of the input characters in hexadecimal format.

  Return type: varchar

  Example:

```
gaussdb=# SELECT rawtohex('1234567');
    rawtohex
----------------
 31323334353637
(1 row)
```

- rawtohex2(any)

  Description: Converts a string in binary format into hexadecimal format.

  The result is the ACSII code of the input characters in hexadecimal format. The slash (/) can be parsed as a normal character and is not escaped.

  When the input is null, the output is also null.

  Return type: text

  Example:

```
set a_format_version='10c';
SET
gaussdb=# set a_format_dev_version='s2';
SET
gaussdb=# select rawtohex2('12\n?$\123/2');
        rawtohex2
--------------------------
 31325C6E3F245C3132332F32
(1 row)
```

- bit2coding(text)

  Description: Reads two-byte characters, saves them based on the little-endian storage logic, parses each character into a hexadecimal ASCII code value, and converts the value into a decimal number.

  Return type: int

  Example:

```
gaussdb=# set a_format_version='10c';
SET
gaussdb=# set a_format_dev_version='s2';
SET
gaussdb=# select bit2coding('1234567890');
 bit2coding
------------
   12849
(1 row)
```

- bit4coding(text)

  Description: Reads four-byte characters, saves them based on the little-endian storage logic, parses each character into a hexadecimal ASCII code value, and converts the value into a decimal number.

  Return type: int

  Example:

```
gaussdb=# set a_format_version='10c';
SET
gaussdb=# set a_format_dev_version='s2';
```



```
SET
gaussdb=# SELECT  bit4coding('1234567890');
 bit4coding
------------
 875770417
(1 row)
```

- to_blob(raw)

  Description: Converts the RAW type to the BLOB type.

  Return type: BLOB

  Example:

  ```
  gaussdb=# SELECT to_blob('0AADD343CDBBD'::RAW(10));
     to_blob
  ----------------
   00AADD343CDBBD
  (1 row)
  ```

  NOTE

  The **to_blob** function is supported only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2**.

- to_bigint(varchar)

  Description: Converts the character type to the bigint type.

  Return type: bigint

  Example:

  ```
  gaussdb=# SELECT to_bigint('123364545554455');
     to_bigint
  ----------------
   123364545554455
  (1 row)
  ```

- to_binary_double(expr)

  Description: Converts an expression to a value of the float8 type.

  expr: supports the number, float4, and float8 data types and character strings that can be implicitly converted to numeric types.

  Return type: float8

  Example:

  ```
  gaussdb=# SELECT to_binary_double('12345678');
   to_binary_double
  ------------------
          12345678
  (1 row)
  ```

  NOTE

  The **to_binary_double** function is supported only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2**.

- to_binary_double(expr, fmt)

  Description: Converts an expression to a value of the float8 type after format matching.

  expr/fmt: supports character strings of the char, nchar, varchar2, and nvarchar2 types. The **expr** also supports numeric types that can be implicitly converted to character string types.

  Return type: float8

  Example:

```
gaussdb=# SELECT to_binary_double('1,2,3', '9,9,9');
 to_binary_double
------------------
              123
(1 row)
```

NOTE

The **to_binary_double** function is supported only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2**.

- to_binary_double(expr default return_value on conversion error)

  Description: Converts an expression to a value of the float8 type. If the conversion fails, the default value **return_value** is returned.

  expr: supports the number, float4, and float8 data types and numeric types that can be implicitly converted to character strings. If **expr** is not of the numeric or character string type, an error message is displayed.

  Return type: float8

  Example:

```
gaussdb=# SELECT to_binary_double(1e2 default 12 on conversion error);
 to_binary_double
------------------
              100
(1 row)

gaussdb=# SELECT to_binary_double('aa' default 12 on conversion error);
 to_binary_double
------------------
               12
(1 row)
```

NOTE

The **to_binary_double** function is supported only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2**.

- to_binary_double(expr default return_value on conversion error, fmt)

  Description: Converts an expression to a value of the float8 type after format matching. If the expression fails to be converted, the default value **return_value** is returned.

  expr/fmt: supports character strings of the char, nchar, varchar2, and nvarchar2 types. The **expr** also supports numeric types that can be implicitly converted to character string types.

  Return type: float8

  Example:

```
gaussdb=# SELECT to_binary_double('12-' default 10 on conversion error, '99S');
 to_binary_double
------------------
             -12
(1 row)

gaussdb=# SELECT to_binary_double('aa-' default 12 on conversion error, '99S');
 to_binary_double
------------------
               12
(1 row)
```

NOTE

The **to_binary_double** function is supported only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2**.

- to_binary_float(expr)

  Description: Converts an expression to a value of the float4 type.

  expr: supports the number, float4, and float8 data types and character strings that can be implicitly converted to numeric types.

  Return type: float4

  Example:

```
gaussdb=# SELECT to_binary_float('12345678');
 to_binary_float
-----------------
     1.23457e+07
(1 row)
```

  NOTE

  The **to_binary_float** function is supported only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2**.

- to_binary_float(expr, fmt)

  Description: Converts an expression to a value of the float4 type after format matching.

  expr/fmt: supports character strings of the char, nchar, varchar2, and nvarchar2 types. The **expr** also supports numeric types that can be implicitly converted to character string types.

  Return type: float4

  Example:

```
gaussdb=# SELECT to_binary_float('1,2,3', '9,9,9');
 to_binary_float
-----------------
             123
(1 row)
```

  NOTE

  The **to_binary_float** function is supported only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2**.

- to_binary_float(expr default return_value on conversion error)

  Description: Converts an expression to a value of the float4 type. If the conversion fails, the default value **return_value** is returned.

  expr: Supports the number, float4, and float8 numeric types and numeric types that can be implicitly converted to character strings. If **expr** is of a non-numeric or non-character string type, an error message is displayed.

  Return type: float4

  Example:

```
gaussdb=# SELECT to_binary_float(1e2 default 12 on conversion error);
 to_binary_float
-----------------
             100
(1 row)

gaussdb=# SELECT to_binary_float('aa' default 12 on conversion error);
 to_binay_float
-----------------
              12
(1 row)
```


> **NOTE**
>
> The **to_binary_float** function is supported only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2**.

- to_binary_float(expr default return_value on conversion error, fmt)

  Description: Converts an expression to a value of the float4 type after format matching. If the expression fails to be converted, the default value **return_value** is returned.

  expr/fmt: supports character strings of the char, nchar, varchar2, and nvarchar2 types. The **expr** also supports numeric types that can be implicitly converted to character string types.

  Return type: float4

  Example:

```
gaussdb=# SELECT to_binary_float('12-' default 10 on conversion error, '99S');
 to_binary_float
-----------------
             -12
(1 row)

gaussdb=# SELECT to_binary_float('aa-' default 12 on conversion error, '99S');
 to_binary_float
-----------------
              12
(1 row)
```

> **NOTE**
>
> The **to_binary_float** function is supported only when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s2**.

- to_char(datetime/interval [, fmt])

  Description: Converts a DATETIME or INTERVAL value of the DATE/TIMESTAMP/TIMESTAMP WITH TIME ZONE/TIMESTAMP WITH LOCAL TIME ZONE type into the VARCHAR type according to the format specified by **fmt**.

  – The optional parameter **fmt** allows for the following types: date, time, week, quarter, and century. Each type has a unique template. The templates can be combined together. Common templates include HH, MI, SS, YYYY, MM, and DD. For details, see **Table 7-56**.

  – A template may have a modification word. FM is a common modification word and is used to suppress the preceding zero or the following blank spaces.

  Return type: varchar

  Example:

```
gaussdb=# SELECT to_char(current_timestamp,'HH12:MI:SS');
 to_char
----------
 10:19:26
(1 row)
gaussdb=# SELECT to_char(current_timestamp,'FMHH12:FMMI:FMSS');
 to_char
----------
 10:19:46
(1 row)
```

- to_char(double precision/real, text)

  Description: Converts the values of the floating point type into the strings in the specified format.

Return type: text

Example:

```
gaussdb=# SELECT to_char(125.8::real, '999D99');
 to_char
---------
  125.80
(1 row)
```

- to_char(numeric/smallint/integer/bigint/double precision/real[, fmt])

  Descriptions: Converts an integer or a value in floating point format into a string in specified format.

  - The optional parameter *fmt* allows for the following types: decimal point, group (thousand) separator, positive/negative sign and currency sign. Each type has a unique template. The templates can be combined together. Common templates include: 9, 0, millesimal sign (,), and decimal point (.). For details, see **Table 7-58**.
  - A template can have a modification word, similar to FM. However, FM does not suppress 0 which is output according to the template.
  - Use the template X or x to convert an integer value into a string in hexadecimal format.

  Return type: varchar

  Example:

```
gaussdb=# SELECT to_char(1485,'9,999');
 to_char
---------
  1,485
(1 row)
gaussdb=# SELECT to_char( 1148.5,'9,999.999');
  to_char
------------
  1,148.500
(1 row)
gaussdb=# SELECT to_char(148.5,'990999.909');
   to_char
-------------
    0148.500
(1 row)
gaussdb=# SELECT to_char(123,'XXX');
 to_char
---------
   7B
(1 row)
```

Table 7-58 fmt parameter of the number type

Pattern	Description
, (comma)	Group (thousand) separator
. (period)	Decimal point
$	$ output to the specified position
0	Value with leading zeros
9	Value with specified digits

Pattern	Description
B	Space returned when the integer part is 0
C	Currency symbol (depending on the locale setting)
D	Decimal point (depending on the locale setting)
EEEE	Scientific notation
G	Group separator (depending on the locale setting)
L	Currency symbol (depending on the locale setting)
MI	Minus sign in the specified position (if the number is less than 0)
PR	Negative value in angle brackets
RN	Roman numeral (ranging from 1 to 3999)
S	Signed number (depending on the locale setting)
TM	Standard number in scientific notation
TM9	Standard number in scientific notation
TME	Standard number in scientific notation
U	Currency symbol (depending on the locale setting)
V	Decimal with specified number of digits shifted
PL	Plus sign in the specified position (if the number is greater than 0)
SG	Plus or minus sign in the specified position
TH or th	Ordinal number suffix

> **NOTE**
>
> The *fmt* parameter can be set to **$**, **C**, **TM**, **TM9**, **TME** or **U** only when **a_format_version** and **a_format_dev_version** are set to **10c** and **s1**, respectively. In addition, the **fmt** parameter cannot be set to **TH**, **PL**, or **SG** in this case.

- to_char(interval, text)

  Description: Converts the values of the time interval type into the strings in the specified format.

  Return type: text

  Example:

  ```
  gaussdb=# SELECT to_char(interval '15h 2m 12s', 'HH24:MI:SS');
   to_char
  ----------
   15:02:12
  (1 row)
  ```

- to_char(integer, text)

  Description: Converts the values of the integer type into the strings in the specified format.

  Return type: text

  Example:

  ```
  gaussdb=# SELECT to_char(125, '999');
   to_char
  ---------
    125
  (1 row)
  ```

- to_char(set)

  Description: Converts a value of the SET type to a string.

  Return value: text

  Example:

  ```
  -- site is a column of the SET type in the employee table.
  gaussdb=# SELECT to_char(site) from employee;
       to_char
  -----------------
   beijing,nanjing
   beijing,wuhan
  (2 rows)
  ```

- to_char(numeric, text)

  Description: Converts the values of the numeric type into the strings in the specified format.

  Return type: text

  Example:

  ```
  gaussdb=# SELECT to_char(-125.8, '999D99S');
   to_char
  ---------
   125.80-
  (1 row)
  ```

- to_char(string)

  Description: Converts the CHAR/VARCHAR/VARCHAR2/CLOB type into the VARCHAR type.

  If this function is used to convert data of the CLOB type, and the value to be converted exceeds the value range of the target type, an error is returned.

Return type: varchar

Example:

```
gaussdb=# SELECT to_char('01110');
 to_char
---------
 01110
(1 row)
```

- to_char(timestamp, text)

  Description: Converts the values of the timestamp type into the strings in the specified format.

  Return type: text

  Example:

  ```
  gaussdb=# SELECT to_char(current_timestamp, 'HH12:MI:SS');
   to_char
  ----------
   10:55:59
  (1 row)
  ```

  NOTE

  - The to_char function outputs the incorrect format (**fmt**) without change. For example, if the **fmt** is **FF10**, FF1 is matched for formatted output, and then **0** is output without change.
  - When **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1**, the to_char function reports an error for the incorrect format (**fmt**).

- to_nchar (datetime/interval [, fmt])

  Description: Converts a DATETIME or INTERVAL value of the DATE/TIMESTAMP/TIMESTAMP WITH TIME ZONE/TIMESTAMP WITH LOCAL TIME ZONE type into the TEXT type according to the format specified by **fmt**.

  – The optional parameter **fmt** allows for the following types: date, time, week, quarter, and century. Each type has a unique template. The templates can be combined together. Common templates include HH, MI, SS, YYYY, MM, and DD. For details, see **Table 7-58**.

  – A template may have a modification word. FM is a common modification word and is used to suppress the preceding zero or the following blank spaces.

  Return type: text

  Example:

  ```
  gaussdb=# SELECT to_nchar(current_timestamp,'HH12:MI:SS');
   to_nchar
  ----------
   10:19:26
  (1 row)
  gaussdb=# SELECT to_nchar(current_timestamp,'FMHH12:FMMI:FMSS');
   to_nchar
  ----------
   10:19:46
  (1 row)
  ```

- to_nchar(double precision/real, text)

  Description: Converts the values of the floating point type into the strings in the specified format.

  Return type: text

  Example:


```
gaussdb=# SELECT to_nchar(125.8::real, '999D99');
 to_nchar
----------
  125.80
(1 row)
```

- to_nchar (numeric/smallint/integer/bigint/double precision/real[, fmt])

  Descriptions: Converts an integer or a value in floating point format into a string in specified format.

  - The optional parameter *fmt* allows for the following types: decimal point, group (thousand) separator, positive/negative sign and currency sign. Each type has a unique template. The templates can be combined together. Common templates include: 9, 0, millesimal sign (,), and decimal point (.). For details, see **Table 7-58**.
  - A template can have a modification word, similar to FM. However, FM does not suppress 0 which is output according to the template.
  - Use the template X or x to convert an integer value into a string in hexadecimal format.

  Return type: text

  Example:

```
gaussdb=# SELECT to_nchar(1485,'9,999');
 to_nchar
----------
  1,485
(1 row)
gaussdb=# SELECT to_nchar( 1148.5,'9,999.999');
  to_nchar
------------
  1,148.500
(1 row)
gaussdb=# SELECT to_nchar(148.5,'990999.909');
   to_nchar
--------------
    0148.500
(1 row)
gaussdb=# SELECT to_nchar(123,'XXX');
 to_nchar
----------
   7B
(1 row)
```

  NOTE

  This function supports the $, C, TM, TM9, TME and U patterns when **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1**. In addition, the **fmt** parameter cannot be set to **TH**, **PL**, or **SG** in this case.

- to_nchar(interval, text)

  Description: Converts the values of the time interval type into the strings in the specified format.

  Return type: text

  Example:

```
gaussdb=# SELECT to_nchar(interval '15h 2m 12s', 'HH24:MI:SS');
 to_nchar
----------
 15:02:12
(1 row)
```

- to_nchar(integer, text)

Description: Converts the values of the integer type into the strings in the specified format.

Return type: text

Example:

```
gaussdb=# SELECT to_nchar(125, '999');
 to_nchar
----------
  125
(1 row)
```

- to_nchar(set)

  Description: Converts a value of the SET type to a string.

  Return value: text

- to_nchar(numeric, text)

  Description: Converts the values of the numeric type into the strings in the specified format.

  Return type: text

  Example:

  ```
  gaussdb=# SELECT to_nchar(-125.8, '999D99S');
   to_nchar
  ----------
   125.80-
  (1 row)
  ```

- to_nchar (string)

  Description: Converts the CHAR/VARCHAR/VARCHAR2/CLOB type into the TEXT type.

  If this function is used to convert data of the CLOB type, and the value to be converted exceeds the value range of the target type, an error is returned.

  Return type: text

  Example:

  ```
  gaussdb=# SELECT to_nchar('01110');
   to_nchar
  ----------
   01110
  (1 row)
  ```

- to_nchar(timestamp, text)

  Description: Converts the values of the timestamp type into the strings in the specified format.

  Return type: text

  Example:

  ```
  gaussdb=# SELECT to_nchar(current_timestamp, 'HH12:MI:SS');
   to_nchar
  -----------
   10:55:59
  (1 row)
  ```

- to_clob(char/nchar/varchar/varchar2/nvarchar/nvarchar2/text/raw)

  Description: Converts the raw type or text character set type CHAR, NCHAR, VARCHAR, VARCHAR2, NVARCHAR, NVARCHAR2, or TEXT to the CLOB type.

  Return type: clob

  Example:


```
gaussdb=# SELECT to_clob('ABCDEF'::RAW(10));
 to_clob
---------
 ABCDEF
(1 row)
gaussdb=# SELECT to_clob('hello111'::CHAR(15));
 to_clob
----------
 hello111
(1 row)
gaussdb=# SELECT to_clob('gauss123'::NCHAR(10));
 to_clob
----------
 gauss123
(1 row)
gaussdb=# SELECT to_clob('gauss234'::VARCHAR(10));
 to_clob
----------
 gauss234
(1 row)
gaussdb=# SELECT to_clob('gauss345'::VARCHAR2(10));
 to_clob
----------
 gauss345
(1 row)
gaussdb=# SELECT to_clob('gauss456'::NVARCHAR2(10));
 to_clob
----------
 gauss456
(1 row)
gaussdb=# SELECT to_clob('World222!'::TEXT);
  to_clob
-----------
 World222!
(1 row)
```

- to_date(text)

  Description: Converts values of the text type into the timestamp in the specified format. Currently, only the following two formats are supported:

  – Format 1: Date without separators, for example, 20150814. The value must contain the complete year, month, and day.

  – Format 2: Date with separators, for example, 2014-08-14. The separator can be any non-digit character.

  Return type: timestamp without time zone

  Example:

```
gaussdb=# SELECT to_date('2015-08-14');
      to_date
---------------------
 2015-08-14 00:00:00
(1 row)
```

  NOTE

  Case execution environment: The value of **a_format_version** is **10c**, the value of **a_format_dev_version** is **s1**, and the value of **nls_timestamp_format** is **YYYY-MM-DD HH24:MI:SS**.

- to_date(text, text)

  Description: Converts the values of the string type into the dates in the specified format.

  Return type: timestamp without time zone

  Example:


```
gaussdb=# SELECT to_date('05 Dec 2000', 'DD Mon YYYY');
      to_date
---------------------
 2000-12-05 00:00:00
(1 row)
```

- to_date(text [DEFAULT return_value ON CONVERSION ERROR [, fmt]])

  Description: Converts a string *text* into a value of the DATE type according to the format specified by *fmt*. If *fmt* is not specified, **a_format_version** is set to **10c**, and **a_format_dev_version** is set to **s1**, the format specified by **nls_timestamp_format** is used for conversion. Otherwise, the fixed format (fmt = 'yyyy-mm-dd hh24-mi-ss') is used for conversion.

  – *text*: any expression that can be calculated to CHAR/VARCHAR2/NCHAR/NVARCHAR2/TEXT string. If null is entered, null is returned.

  – **DEFAULT** *return_value* **ON CONVERSION ERROR**: This parameter is optional, which can be used to specify the return value when *text* fails to be converted to the DATE type. The value of *return_value* can be an expression or a bound variable that can be converted to the CHAR/VARCHAR2/NCHAR/NVARCHAR2/TEXT type or null. The method of converting *return_value* to the DATE type is the same as that of converting *text* to the DATE type. If *return_value* fails to be converted to the DATE type, an error is reported.

  – *fmt*: This parameter is optional, which specifies the date and time model format of *text*. By default, *text* must comply with the default date format. If *fmt* is set to *J*, *text* must be an integer.

  Return type: timestamp without time zone

  Example:

```
gaussdb=# SELECT to_date('2015-08-14');
      to_date
---------------------
 2015-08-14 00:00:00
(1 row)
gaussdb=# SELECT to_date('05 Dec 2000', 'DD Mon YYYY');
      to_date
---------------------
 2000-12-05 00:00:00
(1 row)
gaussdb=# set a_format_version='10c';
SET
gaussdb=# set a_format_dev_version='s1';
SET
gaussdb=# SHOW nls_timestamp_format;
   nls_timestamp_format
----------------------------
 DD-Mon-YYYY HH:MI:SS.FF AM
(1 row)
gaussdb=# SELECT to_date('12-jan-2022' default '12-apr-2022' on conversion error);
      to_date
---------------------
 2022-01-12 00:00:00
(1 row)
gaussdb=# SELECT to_date('12-ja-2022' default '12-apr-2022' on conversion error);
      to_date
---------------------
 2022-04-12 00:00:00
(1 row)
gaussdb=# SELECT to_date('2022-12-12' default '2022-01-01' on conversion error, 'yyyy-mm-dd');
      to_date
---------------------
 2022-12-12 00:00:00
(1 row)
```


CAUTION

- The DEFAULT *return_value* ON CONVERSION ERROR syntax is supported only when **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1**.
- When **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1**, the system may not report an error if the entered year exceeds 9999. For example, the result of **to_date('99999-12-12', 'yyyy-mm-dd hh24:mi:ss')** is **9999-09-12 12:00:00**. If the value of year exceeds 9999, the number following 9999 will be parsed as the next **fmt**. This restriction also applies to **to_timestamp**.

- to_number (expr [, fmt])

  Description: Converts **expr** into a value of the NUMBER type according to the specified format.

  For details about the type conversion formats, see **Table 7-60**.

  If a hexadecimal string is converted into a decimal number, the hexadecimal string can include a maximum of 16 bytes if it is to be converted into a sign-free number.

  During the conversion from a hexadecimal string to a decimal digit, the format string cannot have a character other than x or X. Otherwise, an error is reported.

  Return type: number

  Example:

```
gaussdb=# SELECT to_number('12,454.8-', '99G999D9S');
 to_number
-----------
  -12454.8
(1 row)
```

- to_number(text, text)

  Description: Converts the values of the string type into the numbers in the specified format.

  Return type: numeric

  Example:

```
gaussdb=# SELECT to_number('12,454.8-', '99G999D9S');
 to_number
-----------
  -12454.8
(1 row)
```

- to_number(expr [DEFAULT return_value ON CONVERSION ERROR [, fmt]])

  Description: Converts a string *expr* to a value of the numeric type based on the format specified by *fmt*. If *fmt* is not specified, *text* must be a character string that can be directly converted to numeric, for example, '123', '1e2'.

  For details about the type conversion formats, see **Table 7-61**.

  - **expr**: an expression that can be converted into a CHAR/VARCHAR2/NCHAR/NVARCHAR2/TEXT/INT/FLOAT string. If null is entered, null is returned.
  - **DEFAULT** *return_value* **ON CONVERSION ERROR**: This parameter is optional, which can be used to specify the return value when *expr* fails to

be converted to the numeric type. Similar to *expr*, *return_value* can be any type that can be converted to a character string. Similar to *expr*, *return_value* is converted based on the format specified by *fmt*. The system checks whether *return_value* fails to be converted. If *return_value* fails to be converted, the function reports an error.

– *fmt*: This parameter is optional, which specifies the conversion format of *expr*.

If any input parameter is null, null is returned.

Return type: numeric

Example:

```
gaussdb=# set a_format_version='10c';
gaussdb=# set a_format_dev_version='s1';

gaussdb=# SELECT to_number('1e2');
 to_number
-----------
       100
(1 row)

gaussdb=# SELECT to_number('123.456');
 to_number
-----------
   123.456
(1 row)

gaussdb=# SELECT to_number('123', '999');
 to_number
-----------
       123
(1 row)

gaussdb=# SELECT to_number('123-', '999MI');
 to_number
-----------
      -123
(1 row)

gaussdb=# SELECT to_number('123' default '456-' on conversion error, '999MI');
 to_number
-----------
      -456
(1 row)
```

NOTE

The DEFAULT *return_value* ON CONVERSION ERROR syntax is supported only when **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1**.

- to_timestamp(double precision)

  Description: Converts a Unix century into a timestamp.

  Return type: timestamp with time zone

  Example:

```
gaussdb=# SELECT to_timestamp(1284352323);
      to_timestamp
------------------------
 2010-09-13 12:32:03+08
(1 row)
```

- to_timestamp(string [,fmt])

  Description: Converts a string into a value of the timestamp type according to the format specified by **fmt**. When **fmt** is not specified, perform the conversion according to the format specified by **nls_timestamp_format**.

In **to_timestamp** in GaussDB:

- If the input year *YYYY* is 0, an error will be reported.
- If the input year *YYYY* is less than 0, specify *SYYYY* in **fmt**. The year with the value of n (an absolute value) BC will be output correctly.

Characters in the **fmt** must match the schema for formatting the data and time. Otherwise, an error is reported.

Return type: timestamp without time zone

Example:

```
gaussdb=# SHOW nls_timestamp_format;
   nls_timestamp_format
----------------------------
 DD-Mon-YYYY HH:MI:SS.FF AM
(1 row)

gaussdb=# SELECT to_timestamp('12-sep-2014');
   to_timestamp
---------------------
 2014-09-12 00:00:00
(1 row)
gaussdb=# SELECT to_timestamp('12-Sep-10 14:10:10.123000','DD-Mon-YY HH24:MI:SS.FF');
     to_timestamp
-------------------------
 2010-09-12 14:10:10.123
(1 row)
gaussdb=# SELECT to_timestamp('-1','SYYYY');
     to_timestamp
------------------------
 0001-01-01 00:00:00 BC
(1 row)
gaussdb=# SELECT to_timestamp('98','RR');
   to_timestamp
---------------------
 1998-01-01 00:00:00
(1 row)
gaussdb=# SELECT to_timestamp('01','RR');
   to_timestamp
---------------------
 2001-01-01 00:00:00
(1 row)
```

NOTE

1. When **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1**, **fmt** supports FF[7-9]. When FF[7-9] is used, and the length of the corresponding position in the string is less than or equal to the number following FF, then, the number can be converted. However, the maximum length of the final conversion result is six digits.
2. The result returned by the current_timestamp function cannot be used as a string parameter.

- to_timestamp(text [DEFAULT return_value ON CONVERSION ERROR [, fmt]])

  Description: Converts a string *text* into a value of the DATE type according to the format specified by *fmt*. If *fmt* is not specified, **a_format_version** is set to **10c**, and **a_format_dev_version** is set to **s1**, the format specified by **nls_timestamp_format** is used for conversion. Otherwise, the fixed format (fmt = 'yyyy-mm-dd hh24-mi-ss') is used for conversion.

  - *text*: any expression that can be calculated to CHAR/VARCHAR2/NCHAR/NVARCHAR2/TEXT string. If null is entered, null is returned.
  - **DEFAULT** *return_value* **ON CONVERSION ERROR**: This parameter is optional, which can be used to specify the return value when *text* fails to

be converted to the DATE type. The value of *return_value* can be an expression or a bound variable that can be converted to the CHAR/VARCHAR2/NCHAR/NVARCHAR2/TEXT type or null. The method of converting *return_value* to the timestamp type is the same as that of converting *text* to the timestamp type. If *return_value* fails to be converted to the timestamp type, an error is reported.

- *fmt*: This parameter is optional, which specifies the date and time model format of *text*. By default, *text* must comply with the default date format. If *fmt* is set to *J*, *text* must be an integer.

Return type: timestamp without time zone

Example:

```
gaussdb=# set a_format_version='10c';
SET
gaussdb=# set a_format_dev_version='s1';
SET
gaussdb=# SELECT to_timestamp('11-Sep-11' DEFAULT '12-Sep-10 14:10:10.123000' ON
CONVERSION ERROR,'DD-Mon-YY HH24:MI:SS.FF');
   to_timestamp
---------------------
 2011-09-11 00:00:00
(1 row)
gaussdb=# SELECT to_timestamp('12-Sep-10 14:10:10.123000','DD-Mon-YY HH24:MI:SSXFF');
     to_timestamp
-------------------------
 2010-09-12 14:10:10.123
(1 row)
```

NOTE

The DEFAULT *return_value* ON CONVERSION ERROR syntax is supported only when **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1**.

- to_timestamp(text, text)

  Description: Converts values of the string type into the timestamp of the specified type.

  Return type: timestamp

  Example:

  ```
  gaussdb=# SELECT to_timestamp('05 Dec 2000', 'DD Mon YYYY');
     to_timestamp
  ---------------------
   2000-12-05 00:00:00
  (1 row)
  ```

- to_timestamp_tz(string [DEFAULT return_value ON CONVERSION ERROR] [,fmt])

  Description: Converts a string into a value of the timestamp type with the time zone according to the format specified by **fmt**. When **fmt** is not specified, perform the conversion according to the format specified by **nls_timestamp_tz_format**.

  - **DEFAULT** *return_value* **ON CONVERSION ERROR**: This parameter is optional. If *string* fails to be converted to the timestamp type with time zone, *return_value* is converted to the timestamp type with time zone.
  - *fmt*: This parameter is optional, which specifies the date and time model format of *string*. The setting of this parameter is the same as that in the to_timestamp function.

  Return type: timestamp with time zone

Example:

```
gaussdb=# SELECT to_timestamp_tz('05 DeX 2000' DEFAULT '05 Dec 2001' ON CONVERSION ERROR,
'DD Mon YYYY');
    to_timestamp_tz
---------------------------
 2001-12-05 00:00:00+08:00
(1 row)
```

NOTE

This function is valid only when **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1**.

- to_timestamp_tz(string [DEFAULT return_value ON CONVERSION ERROR], fmt, nlsparam)

  Description: Converts a string into a value of the timestamp type with the time zone according to the format specified by **fmt**. If *string* fails to be converted to the timestamp type with time zone, *return_value* is converted to the timestamp type with time zone. **nlsparam** specifies the language of the month and day in the time character string. The format is **'nls_date_language=language'**. Currently, **language** supports only **ENGLISH** and **AMERICAN**. Currently, the result of correctly using the **nlsparam** parameter is the same as that of omitting the **nlsparam** parameter. For details, see **Table 7-59**.

  Return type: timestamp with time zone

Table 7-59 Parameters

Parameter	Type	Description
string	text	Converts to a string of the timestamp with time zone type.
return_value	text	If a string fails to be converted to the timestamp type with time zone, **return_value** is converted to the timestamp type with time zone.
fmt	text	Specifies the date and time model format of the **string** parameter.
nlsparam	text	Specifies the language of the month and day in the **string** parameter.

Example:

```
gaussdb=# SELECT to_timestamp_tz('05 DeX 2000' DEFAULT '05 Dec 2001' ON CONVERSION ERROR,
'DD Mon YYYY','nls_date_language=AMERICAN');
    to_timestamp_tz
---------------------------
 2001-12-05 00:00:00+08:00
(1 row)
```

NOTE

his function is valid only when **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s4.**

- to_dsinterval(text)

  Description: Converts characters to the interval type. SQL-compatible and ISO formats are supported.

  Return type: interval

  Example:

```
gaussdb=# SELECT to_dsinterval('12 1:2:3.456');
    to_dsinterval
----------------------
 12 days 01:02:03.456
(1 row)

gaussdb=# SELECT to_dsinterval('P3DT4H5M6S');
  to_dsinterval
-----------------
 3 days 04:05:06
(1 row)
```

  NOTE

  This function is valid only when **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s2**.

- to_yminterval(text)

  Description: Converts characters to the interval type. SQL-compatible and ISO formats are supported.

  Return type: interval

  Example:

```
gaussdb=# SELECT to_yminterval('1-1');
 to_yminterval
-----------------
 1 year 1 mon
(1 row)

gaussdb=# SELECT to_yminterval('P13Y3M4DT4H2M5S');
 to_yminterval
-----------------
 13 years 3 mons
(1 row)
```

  NOTE

  This function is valid only when **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s2**.

Table 7-60 Template patterns for numeric formatting

Pattern	Description
9	Value with specified digits
0	Values with leading zeros
Period (.)	Decimal point
Comma (,)	Group (thousand) separator
PR	Negative values in angle brackets
S	Signed number (depending on the locale setting)

Pattern	Description
L	Currency symbol (depending on the locale setting)
D	Decimal point (depending on the locale setting)
G	Group separator (depending on the locale setting)
MI	Minus sign in the specified position (if the number is less than 0)
PL	Plus sign in the specified position (if the number is greater than 0)
SG	Plus or minus sign in the specified position
RN	Roman numeral (ranging from 1 to 3999)
TH or th	Ordinal number suffix
V	Decimal with specified number of digits shifted
x or X	Hexadecimal-to-decimal conversion identifier

Table 7-61 Template patterns for to_number formatting

Pattern	Description
9	Matches a digit. The number of digits "9" can be greater than or equal to that in the corresponding position in *expr*.
0	Strictly matches a digit. The number of digits "0" must be equal to that in *expr*.
5	Matches digit 0 or 5.
Period (.)	Decimal point in the specified position.
Comma (,)	Group (thousand) separator in the specified position. Multiple commas can be specified in *fmt*.
B	No leading sign.
PR	A negative value in angle brackets, or a positive value with no leading sign.
S	A negative value with the leading minus sign (–) or a positive value with the leading plus sign (+).
MI	A negative value with the leading minus sign (–) or a positive value with no leading sign.
$	Leading dollar sign.
L	Local currency symbol.

Pattern	Description
C	Symbol of currency in the specified position (complying with the ISO standard).
U	Dual-currency symbol.
D	Decimal point (depending on the locale setting).
G	Group separator (complying with the ISO standard). Multiple commas can be specified in *fmt*.
RN / rn	Roman numeral (ranging from 1 to 3999). to_number does not support this format.
V	to_number does not support this format.
X / x	Conversion between hexadecimal and decimal.
TM	to_number does not support this format.
FM	This format can be used only at the beginning of *fmt*.
EEEE	Conversion based on the scientific notation model.

NOTE

When **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1**, refer to this table for the setting of *fmt*. Otherwise, refer to the previous table. The *fmt* setting complying with the ISO standard is affected by the values of **LC_MONETARY** and **LC_NUMERIC** parameters.

- cast_varchar2_to_raw_for_histogram(varchar2)
  Description: Converts from the varchar2 type to the raw type.
  Return type: raw
- abstime_text(abstime)
  Description: Converts abstime to text.
  Parameter: abstime
  Return type: text
- abstime_to_smalldatetime(abstime)
  Description: Converts abstime to smalldatetime.
  Parameter: abstime
  Return type: smalldatetime
- bigint_tid(bigint)
  Description: Converts bigint to tid.
  Parameter: bigint
  Return type: tid
- bool_int1(boolean)
  Description: Converts Boolean to int1.
  Parameter: Boolean

Return type: tinyint

- bool_int2(boolean)
  Description: Converts Boolean to int2.
  Parameter: Boolean
  Return type: smallint
- bool_int8(boolean)
  Description: Converts Boolean to int8.
  Parameter: Boolean
  Return type: bigint
- bpchar_date(character)
  Description: Converts a string to a date.
  Parameter: character
  Return type: date
- bpchar_float4(character)
  Description: Converts a string to float4.
  Parameter: character
  Return type: real
- bpchar_float8(character)
  Description: Converts a string to float8.
  Parameter: character
  Return type: double precision
- bpchar_int4(character)
  Description: Converts a string to int4.
  Parameter: character
  Return type: integer
- bpchar_int8(character)
  Description: Converts a string to int8.
  Parameter: character
  Return type: bigint
- bpchar_numeric(character)
  Description: Converts a string to numeric.
  Parameter: character
  Return type: numeric
- bpchar_timestamp(character)
  Description: Converts a string to a timestamp.
  Parameter: character
  Return type: timestamp without time zone
- bpchar_to_smalldatetime(character)
  Description: Converts a string to smalldatetime.
  Parameter: character
  Return type: smalldatetime

- complex_array_in(cstring, oid, int2vector)
  Description: Converts the external complex_array type to the internal anyarray array type.
  Parameter: cstring, oid, int2vector
  Return type: anyarray
- date_bpchar(date)
  Description: Converts the date type to bpchar.
  Parameter: date
  Return type: character
- date_text(date)
  Description: Converts date to text.
  Parameter: date
  Return type: text
- date_varchar(date)
  Description: Converts date to varchar.
  Parameter: date
  Return type: character varying
- f4toi1(real)
  Description: Forcibly converts float4 to uint8.
  Parameter: real
  Return type: tinyint
- f8toi1(double precision)
  Description: Forcibly converts float8 to uint8.
  Parameter: double precision
  Return type: tinyint
- float4_bpchar(real)
  Description: Converts float4 to bpchar.
  Parameter: real
  Return type: character
- float4_text(real)
  Description: Converts float4 to text.
  Parameter: real
  Return type: text
- float4_varchar(real)
  Description: Converts float4 to varchar.
  Parameter: real
  Return type: character varying
- float8_bpchar(double precision)
  Description: Converts float8 to bpchar.
  Parameter: double precision
  Return type: character

- float8_interval(double precision)
  Description: Converts float8 to interval.
  Parameter: double precision
  Return type: interval
- float8_text(double precision)
  Description: Converts float8 to text.
  Parameter: double precision
  Return type: text
- float8_varchar(double precision)
  Description: Converts float8 to varchar.
  Parameter: double precision
  Return type: character varying
- i1tof4(tinyint)
  Description: Converts uint8 to float4.
  Parameter: tinyint
  Return type: real
- i1tof8(tinyint)
  Description: Converts uint8 to float8.
  Parameter: tinyint
  Return type: double precision
- i1toi2(tinyint)
  Description: Converts uint8 to int16.
  Parameter: tinyint
  Return type: smallint
- i1toi4(tinyint)
  Description: Converts uint8 to int32.
  Parameter: tinyint
  Return type: integer
- i1toi8(tinyint)
  Description: Converts uint8 to int64.
  Parameter: tinyint
  Return type: bigint
- i2toi1(smallint)
  Description: Converts int16 to uint8.
  Parameter: smallint
  Return type: tinyint
- i4toi1(integer)
  Description: Converts int32 to uint8.
  Parameter: integer
  Return type: tinyint

- i8toi1(bigint)

  Description: Converts int64 to uint8.

  Parameter: bigint

  Return type: tinyint
- int1_avg_accum(bigint[], tinyint)

  Description: Adds the second parameter of the uint8 type to the first parameter. The first parameter is an array of the bigint type.

  Parameter: bigint[], tinyint

  Return type: bigint[]
- int1_bool(tinyint)

  Description: Converts uint8 to Boolean.

  Parameter: tinyint

  Return type: Boolean
- int1_bpchar(tinyint)

  Description: Converts uint8 to bpchar.

  Parameter: tinyint

  Return type: character
- int1_mul_cash(tinyint, money)

  Description: Returns the product of a parameter of the int8 type and a parameter of the cash type. The return type is cash.

  Parameter: tinyint, money

  Return type: money
- int1_numeric(tinyint)

  Description: Converts uint8 to numeric.

  Parameter: tinyint

  Return type: numeric
- int1_nvarchar2(tinyint)

  Description: Converts uint8 to nvarchar2.

  Parameter: tinyint

  Return type: nvarchar2
- int1_text(tinyint)

  Description: Converts uint8 to text.

  Parameter: tinyint

  Return type: text
- int1_varchar(tinyint)

  Description: Converts uint8 to varchar.

  Parameter: tinyint

  Return type: character varying
- int1in(cstring)

  Description: Converts a string into an unsigned 1-byte integer.

  Parameter: cstring

Return type: tinyint

- int1out(tinyint)
  Description: Converts an unsigned 1-byte integer into a string.
  Parameter: tinyint
  Return type: cstring
- int1up(tinyint)
  Description: Converts an input integer to an unsigned 1-byte integer.
  Parameter: tinyint
  Return type: tinyint
- int2_bool(smallint)
  Description: Converts a signed two-byte integer to the bool type.
  Parameter: smallint
  Return type: Boolean
- int2_bpchar(smallint)
  Description: Converts a signed two-byte integer to the bpchar type.
  Parameter: smallint
  Return type: character
- int2_text(smallint)
  Description: Converts a signed two-byte integer to the text type.
  Parameter: smallint
  Return type: text
- int2_varchar(smallint)
  Description: Converts a signed two-byte integer to the varchar type.
  Parameter: smallint
  Return type: character varying
- int4_bpchar(integer)
  Description: Converts a signed four-byte integer to bpchar.
  Parameter: integer
  Return type: character
- int4_text(integer)
  Description: Converts a signed four-byte integer to the text type.
  Parameter: integer
  Return type: text
- int4_varchar(integer)
  Description: Converts a signed four-byte integer into varchar.
  Parameter: integer
  Return type: character varying
- int8_bool(bigint)
  Description: Converts an eight-byte signed integer to a Boolean value.
  Parameter: bigint
  Return type: Boolean

- int8_bpchar(bigint)
  Description: Converts an 8-byte signed integer to bpchar.
  Parameter: bigint
  Return type: character
- int8_text(bigint)
  Description: Converts an eight-byte signed integer to the text type.
  Parameter: bigint
  Return type: text
- int8_varchar(bigint)
  Description: Converts an eight-byte signed integer to varchar.
  Parameter: bigint
  Return type: character varying
- intervaltonum(interval)
  Description: Converts the internal dats type date to numeric.
  Parameter: interval
  Return type: numeric
- numeric_bpchar(numeric)
  Description: Converts numeric to bpchar.
  Parameter: numeric
  Return type: character
- numeric_int1(numeric)
  Description: Converts numeric to a signed one-byte integer.
  Parameter: numeric
  Return type: tinyint
- numeric_text(numeric)
  Description: Converts numeric to text.
  Parameter: numeric
  Return type: text
- numeric_varchar(numeric)
  Description: Converts numeric to varchar.
  Parameter: numeric
  Return type: character varying
- nvarchar2in(cstring, oid, integer)
  Description: Converts c string to varchar.
  Parameter: cstring, oid, integer
  Return type: nvarchar2
- nvarchar2out(nvarchar2)
  Description: Converts text into a c string.
  Parameter: nvarchar2
  Return type: cstring

- nvarchar2send(nvarchar2)
  Description: Converts varchar to binary.
  Parameter: nvarchar2
  Return type: bytea
- oidvectorin_extend(cstring)
  Description: Converts a string to oidvector.
  Parameter: cstring
  Return type: oidvector_extend
- oidvectorout_extend(oidvector_extend)
  Description: Converts oidvector to a string.
  Parameter: oidvector_extend
  Return type: cstring
- oidvectorsend_extend(oidvector_extend)
  Description: Converts oidvector to a string.
  Parameter: oidvector_extend
  Return type: bytea
- reltime_text(reltime)
  Description: Converts reltime to text.
  Parameter: reltime
  Return type: text
- text_date(text)
  Description: Converts the text type to the date type.
  Parameter: text
  Return type: date
- text_float4(text)
  Description: Converts text to float4.
  Parameter: text
  Return type: real
- text_float8(text)
  Description: Converts the text type to float8.
  Parameter: text
  Return type: double precision
- text_int1(text)
  Description: Converts the text type to int1.
  Parameter: text
  Return type: tinyint
- text_int2(text)
  Description: Converts the text type to the int2 type.
  Parameter: text
  Return type: smallint

- text_int4(text)
  Description: Converts the text type to int4.
  Parameter: text
  Return type: integer
- text_int8(text)
  Description: Converts the text type to the int8 type.
  Parameter: text
  Return type: bigint
- text_numeric(text)
  Description: Converts the text type to the numeric type.
  Parameter: text
  Return type: numeric
- text_timestamp(text)
  Description: Converts the text type to the timestamp type.
  Parameter: text
  Return type: timestamp without time zone
- time_text(time without time zone)
  Description: Converts the time type to the text type.
  Parameter: time without time zone
  Return type: text
- timestamp_text(timestamp without time zone)
  Description: Converts the timestamp type to the text type.
  Parameter: timestamp without time zone
  Return type: text
- timestamp_to_smalldatetime(timestamp without time zone)
  Description: Converts the timestamp type to the smalldatetime type.
  Parameter: timestamp without time zone
  Return type: smalldatetime
- timestamp_varchar(timestamp without time zone)
  Description: Converts the timestamp type to varchar.
  Parameter: timestamp without time zone
  Return type: character varying
- timestamptz_to_smalldatetime(timestamp with time zone)
  Description: Converts timestamptz to smalldatetime.
  Parameter: timestamp with time zone
  Return type: smalldatetime
- timestampzone_text(timestamp with time zone)
  Description: Converts the timestampzone type to the text type.
  Parameter: timestamp with time zone
  Return type: text

- timetz_text(time with time zone)
  Description: Converts the timetz type to the text type.
  Parameter: time with time zone
  Return type: text
- to_integer(character varying)
  Description: Converts data to the integer type.
  Parameter: character varying
  Return type: integer
- to_interval(character varying)
  Description: Converts to the interval type.
  Parameter: character varying
  Return type: interval
- to_numeric(character varying)
  Description: Converts to the numeric type.
  Parameter: character varying
  Return type: numeric
- to_nvarchar2(numeric)
  Description: Converts to the nvarchar2 type.
  Parameter: numeric
  Return type: nvarchar2
- to_text(smallint)
  Description: Converts to the text type.
  Parameter: smallint
  Return type: text
- to_ts(character varying)
  Description: Converts to the ts type.
  Parameter: character varying
  Return type: timestamp without time zone
- to_varchar2(timestamp without time zone)
  Description: Converts to the varchar2 type.
  Parameter: timestamp without time zone
  Return type: character varying
- varchar_date(character varying)
  Description: Converts varchar to date.
  Parameter: character varying
  Return type: date
- varchar_float4(character varying)
  Description: Converts varchar to float4.
  Parameter: character varying
  Return type: real

- varchar_float8(character varying)
  Description: Converts the varchar type to the float8 type.
  Parameter: character varying
  Return type: double precision
- varchar_int4(character varying)
  Description: Converts the type from varchar to int4.
  Parameter: character varying
  Return type: integer
- varchar_int8(character varying)
  Description: Converts the varchar type to the int8 type.
  Parameter: character varying
  Return type: bigint
- varchar_numeric(character varying)
  Description: Converts varchar to numeric.
  Parameter: character varying
  Return type: numeric
- varchar_timestamp(character varying)
  Description: Converts varchar to timestamp.
  Parameter: character varying
  Return type: timestamp without time zone
- varchar2_to_smlldatetime(character varying)
  Description: Converts varchar2 to smlldatetime.
  Parameter: character varying
  Return type: smalldatetime
- xidout4(xid32)
  Description: The xid output is a four-byte number.
  Parameter: xid32
  Return type: cstring
- xidsend4(xid32)
  Description: Converts xid to the binary format.
  Parameter: xid32
  Return type: bytea
- treat(expr AS [JSON | REF] schema.type)
  Description: Converts expr to the type (JSON or user-defined type) specified by the keyword after AS.
  Return value type: JSON or user-defined type.
  Example:

```
gaussdb=# CREATE TABLE json_doc(data CLOB);
gaussdb=# INSERT INTO json_doc values('{"name":"a"}');
gaussdb=# SELECT treat(data as json) FROM json_doc;
    json
--------------
 {"name":"a"}
```



```
(1 row)
gaussdb=# DROP TABLE json_doc;
DROP TABLE
```

- nesttable_to_array(anynesttable)

  Description: Converts a nest-table collection type to an array type with the same elements.

  Parameter: anynesttable

  Return type: anyarray

  Example:

```
gaussdb=# create or replace procedure p1 is
gaussdb$#     type t1 is table of int;
gaussdb$#     v1 t1 := t1(1, 2, 3);
gaussdb$#     v2 int[] := cast(v1 as int[]);
gaussdb$# begin
gaussdb$#     raise info '%', v2;
gaussdb$# end;
gaussdb$# /
CREATE PROCEDURE
gaussdb=# call p1();
INFO:  {1,2,3}
 p1
----

(1 row)

gaussdb=# CREATE type t1 is table of int;
CREATE TYPE
gaussdb=# SELECT cast(t1(1, 2, 3) as int[]) result;
 result
---------
 {1,2,3}
(1 row)

gaussdb=# DROP procedure p1;
DROP PROCEDURE

gaussdb=# DROP type t1;
DROP TYPE
```

- indexbytableint_to_array(anyindexbytable)

  Description: Converts a collection whose index type is integer to an array type with the same elements.

  Parameter: anyindexbytable

  Return type: anyarray

  Example:

```
gaussdb=# create or replace package pkg1 is
gaussdb$#     type t1 is table of int index by int;
gaussdb$#     procedure p1();
gaussdb$# end pkg1;
gaussdb$# /
CREATE PACKAGE
gaussdb=#
gaussdb=# create or replace package body pkg1 is
gaussdb$#     procedure p1() is
gaussdb$#         v1 t1 := t1(1 => 1, 2 => 2, 3 => 3);
gaussdb$#         v2 int[];
gaussdb$#     begin
gaussdb$#         v2 := cast(v1 as int[]);
gaussdb$#         raise info '%', v2;
gaussdb$#     end;
gaussdb$# end pkg1;
gaussdb$# /
```



```
CREATE PACKAGE BODY
gaussdb=#
gaussdb=# call pkg1.p1();
INFO:  {1,2,3}
 p1
----

(1 row)

gaussdb=# SELECT indexbytableint_to_array(pkg1.t1(1 => 1, 2 => 2, 3 => 3));
 indexbytableint_to_array
--------------------------
 {1,2,3}
(1 row)

gaussdb=# DROP package pkg1;
DROP PACKAGE
```

Encoding Type Conversion

- convert_to_nocase(text, text)

  Description: Converts a string into a specified encoding type.

  Return type: bytea

  Example:

```
gaussdb=# SELECT convert_to_nocase('12345', 'GBK');
 convert_to_nocase
-------------------
 \x3132333435
(1 row)
```

7.6.10 Geometric Functions and Operators

Geometric Operators

- +

  Description: Translation

  Example:

```
gaussdb=# SELECT box '((0,0),(1,1))' + point '(2.0,0)' AS RESULT;
   result
-------------
 (3,1),(2,0)
(1 row)
```

- -

  Description: Translation

  Example:

```
gaussdb=# SELECT box '((0,0),(1,1))' - point '(2.0,0)' AS RESULT;
    result
---------------
 (-1,1),(-2,0)
(1 row)
```

- *

  Description: Scaling out/Rotation

  Example:

```
gaussdb=# SELECT box '((0,0),(1,1))' * point '(2.0,0)' AS RESULT;
   result
-------------
 (2,2),(0,0)
(1 row)
```


- /

  Description: Scaling in/Rotation

  Example:

```
gaussdb=# SELECT box '((0,0),(2,2))' / point '(2.0,0)' AS RESULT;
   result
-------------
 (1,1),(0,0)
(1 row)
```

- #

  Description: Intersection of two figures

  Example:

```
gaussdb=# SELECT box '((1,-1),(-1,1))' # box '((1,1),(-2,-2))' AS RESULT;
 result
---------------
 (1,1),(-1,-1)
(1 row)
```

- #

  Description: Number of paths or polygon vertexes

  Example:

```
gaussdb=# SELECT # path'((1,0),(0,1),(-1,0))' AS RESULT;
 result
--------
      3
(1 row)
```

- @-@

  Description: Length or circumference

  Example:

```
gaussdb=# SELECT @-@ path '((0,0),(1,0))' AS RESULT;
 result
--------
      2
(1 row)
```

- @@

  Description: Center of box

  Example:

```
gaussdb=# SELECT @@ circle '((0,0),10)' AS RESULT;
 result
--------
 (0,0)
(1 row)
```

- <->

  Description: Distance between the two figures

  Example:

```
gaussdb=# SELECT circle '((0,0),1)' <-> circle '((5,0),1)' AS RESULT;
 result
--------
      3
(1 row)
```

- &&

  Description: Overlaps? (One point in common makes this true.)

  Example:

```
gaussdb=# SELECT box '((0,0),(1,1))' && box '((0,0),(2,2))' AS RESULT;
 result
```



```
--------
 t
(1 row)
```

- <<

  Description: Is strictly left of (no common horizontal coordinate)?

  Example:

```
gaussdb=# SELECT circle '((0,0),1)' << circle '((5,0),1)' AS RESULT;
 result
--------
 t
(1 row)
```

- >>

  Description: Is strictly right of (no common horizontal coordinate)?

  Example:

```
gaussdb=# SELECT circle '((5,0),1)' >> circle '((0,0),1)' AS RESULT;
 result
--------
 t
(1 row)
```

- &<

  Description: Does not extend to the right of?

  Example:

```
gaussdb=# SELECT box '((0,0),(1,1))' &< box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- &>

  Description: Does not extend to the left of?

  Example:

```
gaussdb=# SELECT box '((0,0),(3,3))' &> box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- <<|

  Description: Is strictly below (no common horizontal coordinate)?

  Example:

```
gaussdb=# SELECT box '((0,0),(3,3))' <<| box '((3,4),(5,5))' AS RESULT;
 result
--------
 t
(1 row)
```

- |>>

  Description: Is strictly above (no common horizontal coordinate)?

  Example:

```
gaussdb=# SELECT box '((3,4),(5,5))' |>> box '((0,0),(3,3))' AS RESULT;
 result
--------
 t
(1 row)
```

- &<|

  Description: Does not extend above?

Example:

```
gaussdb=# SELECT box '((0,0),(1,1))' &<| box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- |&>

  Description: Does not extend below?

  Example:

```
gaussdb=# SELECT box '((0,0),(3,3))' |&> box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- <^

  Description: Is below (allows touching)?

  Example:

```
gaussdb=# SELECT box '((0,0),(-3,-3))' <^ box '((0,0),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- >^

  Description: Is above (allows touching)?

  Example:

```
gaussdb=# SELECT box '((0,0),(2,2))' >^ box '((0,0),(-3,-3))'  AS RESULT;
 result
--------
 t
(1 row)
```

- ?#

  Description: Intersect?

  Example:

```
gaussdb=# SELECT lseg '((-1,0),(1,0))' ?# box '((-2,-2),(2,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- ?-

  Description: Is horizontal?

  Example:

```
gaussdb=# SELECT ?- lseg '((-1,0),(1,0))' AS RESULT;
 result
--------
 t
(1 row)
```

- ?-

  Description: Are horizontally aligned?

  Example:

```
gaussdb=# SELECT point '(1,0)' ?- point '(0,0)' AS RESULT;
 result
--------
 t
(1 row)
```


- ?|

  Description: Is vertical?

  Example:

```
gaussdb=# SELECT ?| lseg '((-1,0),(1,0))' AS RESULT;
 result
--------
 f
(1 row)
```

- ?|

  Description: Are vertically aligned?

  Example:

```
gaussdb=# SELECT point '(0,1)' ?| point '(0,0)' AS RESULT;
 result
--------
 t
(1 row)
```

- ?-|

  Description: Are perpendicular?

  Example:

```
gaussdb=# SELECT lseg '((0,0),(0,1))' ?-| lseg '((0,0),(1,0))' AS RESULT;
 result
--------
 t
(1 row)
```

- ?||

  Description: Are parallel?

  Example:

```
gaussdb=# SELECT lseg '((-1,0),(1,0))' ?|| lseg '((-1,2),(1,2))' AS RESULT;
 result
--------
 t
(1 row)
```

- @>

  Description: Contains?

  Example:

```
gaussdb=# SELECT circle '((0,0),2)' @> point '(1,1)' AS RESULT;
 result
--------
 t
(1 row)
```

- <@

  Description: Contained in or on?

  Example:

```
gaussdb=# SELECT point '(1,1)' <@ circle '((0,0),2)' AS RESULT;
 result
--------
 t
(1 row)
```

- ~=

  Description: Specifies whether two figures are the same.

  Example:

```
gaussdb=# SELECT polygon '((0,0),(1,1))' ~= polygon '((1,1),(0,0))' AS RESULT;
 result
```



```
--------
 t
(1 row)
```

Geometric Functions

- area(object)

  Description: Area calculation

  Return type: double precision

  Example:

  ```
  gaussdb=# SELECT area(box '((0,0),(1,1))') AS RESULT;
   result
  --------
       1
  (1 row)
  ```

- center(object)

  Description: Figure center calculation

  Return type: point

  Example:

  ```
  gaussdb=# SELECT center(box '((0,0),(1,2))') AS RESULT;
   result
  ---------
   (0.5,1)
  (1 row)
  ```

- diameter(circle)

  Description: Circle diameter calculation

  Return type: double precision

  Example:

  ```
  gaussdb=# SELECT diameter(circle '((0,0),2.0)') AS RESULT;
   result
  --------
       4
  (1 row)
  ```

- height(box)

  Description: Vertical size of box

  Return type: double precision

  Example:

  ```
  gaussdb=# SELECT height(box '((0,0),(1,1))') AS RESULT;
   result
  --------
       1
  (1 row)
  ```

- isclosed(path)

  Description: A closed path?

  Return type: Boolean

  Example:

  ```
  gaussdb=# SELECT isclosed(path '((0,0),(1,1),(2,0))') AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- isopen(path)

Description: An open path?

Return type: Boolean

Example:

```
gaussdb=# SELECT isopen(path '[(0,0),(1,1),(2,0)]') AS RESULT;
 result
--------
 t
(1 row)
```

- length(object)

  Description: Length calculation

  Return type: double precision

  Example:

```
gaussdb=# SELECT length(path '((-1,0),(1,0))') AS RESULT;
 result
--------
      4
(1 row)
```

- npoints(path)

  Description: Number of points in path

  Return type: int

  Example:

```
gaussdb=# SELECT npoints(path '[(0,0),(1,1),(2,0)]') AS RESULT;
 result
--------
      3
(1 row)
```

- npoints(polygon)

  Description: Number of points in polygon

  Return type: int

  Example:

```
gaussdb=# SELECT npoints(polygon '((1,1),(0,0))') AS RESULT;
 result
--------
      2
(1 row)
```

- pclose(path)

  Description: Converts path to closed.

  Return type: path

  Example:

```
gaussdb=# SELECT pclose(path '[(0,0),(1,1),(2,0)]') AS RESULT;
       result
---------------------
 ((0,0),(1,1),(2,0))
(1 row)
```

- popen(path)

  Description: Converts path to open.

  Return type: path

  Example:

```
gaussdb=# SELECT popen(path '((0,0),(1,1),(2,0))') AS RESULT;
       result
---------------------
```



```
 [(0,0),(1,1),(2,0)]
(1 row)
```

- radius(circle)

  Description: Circle diameter calculation

  Return type: double precision

  Example:

```
gaussdb=# SELECT radius(circle '((0,0),2.0)') AS RESULT;
 result
--------
      2
(1 row)
```

- width(box)

  Description: Horizontal size of box

  Return type: double precision

  Example:

```
gaussdb=# SELECT width(box '((0,0),(1,1))') AS RESULT;
 result
--------
      1
(1 row)
```

Geometric Type Conversion Functions

- box(circle)

  Description: Circle to box

  Return type: box

  Example:

```
gaussdb=# SELECT box(circle '((0,0),2.0)') AS RESULT;
                                result
-----------------------------------------------------------------------
 (1.41421356237309,1.41421356237309),(-1.41421356237309,-1.41421356237309)
(1 row)
```

- box(point, point)

  Description: Points to box

  Return type: box

  Example:

```
gaussdb=# SELECT box(point '(0,0)', point '(1,1)') AS RESULT;
   result
-------------
 (1,1),(0,0)
(1 row)
```

- box(polygon)

  Description: Polygon to box

  Return type: box

  Example:

```
gaussdb=# SELECT box(polygon '((0,0),(1,1),(2,0))') AS RESULT;
   result
-------------
 (2,1),(0,0)
(1 row)
```

- circle(box)

  Description: Box to circle

Return type: circle

Example:

```
gaussdb=# SELECT circle(box '((0,0),(1,1))') AS RESULT;
            result
-------------------------------
 <(0.5,0.5),0.707106781186548>
(1 row)
```

- circle(point, double precision)

  Description: Center and radius to circle

  Return type: circle

  Example:

```
gaussdb=# SELECT circle(point '(0,0)', 2.0) AS RESULT;
  result
-----------
 <(0,0),2>
(1 row)
```

- circle(polygon)

  Description: Polygon to circle

  Return type: circle

  Example:

```
gaussdb=# SELECT circle(polygon '((0,0),(1,1),(2,0))') AS RESULT;
                  result
-------------------------------------------
 <(1,0.333333333333333),0.924950591148529>
(1 row)
```

- lseg(box)

  Description: Box diagonal to line segment

  Return type: lseg

  Example:

```
gaussdb=# SELECT lseg(box '((-1,0),(1,0))') AS RESULT;
    result
----------------
 [(1,0),(-1,0)]
(1 row)
```

- lseg(point, point)

  Description: Points to line segment

  Return type: lseg

  Example:

```
gaussdb=# SELECT lseg(point '(-1,0)', point '(1,0)') AS RESULT;
    result
----------------
 [(-1,0),(1,0)]
(1 row)
```

- slope(point, point)

  Description: Calculates the slope of a straight line formed by two points.

  Return type: double

  Example:

```
gaussdb=# SELECT slope(point '(1,1)', point '(0,0)') AS RESULT;
 result
--------
      1
(1 row)
```


- path(polygon)

  Description: Polygon to path

  Return type: path

  Example:

```
gaussdb=# SELECT path(polygon '((0,0),(1,1),(2,0))') AS RESULT;
      result
--------------------
 ((0,0),(1,1),(2,0))
(1 row)
```

- point(double precision, double precision)

  Description: Points

  Return type: point

  Example:

```
gaussdb=# SELECT point(23.4, -44.5) AS RESULT;
   result
--------------
 (23.4,-44.5)
(1 row)
```

- point(box)

  Description: Center of box

  Return type: point

  Example:

```
gaussdb=# SELECT point(box '((-1,0),(1,0))') AS RESULT;
 result
--------
 (0,0)
(1 row)
```

- point(circle)

  Description: Center of circle

  Return type: point

  Example:

```
gaussdb=# SELECT point(circle '((0,0),2.0)') AS RESULT;
 result
--------
 (0,0)
(1 row)
```

- point(lseg)

  Description: Center of a line segment

  Return type: point

  Example:

```
gaussdb=# SELECT point(lseg '((-1,0),(1,0))') AS RESULT;
 result
--------
 (0,0)
(1 row)
```

- point(polygon)

  Description: Center of a polygon

  Return type: point

  Example:

```
gaussdb=# SELECT point(polygon '((0,0),(1,1),(2,0))') AS RESULT;
      result
```



```
----------------------
 (1,0.333333333333333)
(1 row)
```

- polygon(box)

  Description: Box to 4-point polygon

  Return type: polygon

  Example:

```
gaussdb=# SELECT polygon(box '((0,0),(1,1))') AS RESULT;
          result
---------------------------
 ((0,0),(0,1),(1,1),(1,0))
(1 row)
```

- polygon(circle)

  Description: Circle to 12-point polygon

  Return type: polygon

  Example:

```
gaussdb=# SELECT polygon(circle '((0,0),2.0)') AS RESULT;

result

------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------
 ((-2,0),(-1.73205080756888,1),(-1,1.73205080756888),(-1.22464679914735e-16,2),
(1,1.73205080756888),(1.73205080756888,1),(2,2.44929359829471e-16),
(1.73205080756888,-0.999999999999999),(1,-1.73205080756888),(3.67394039744206e-16,-2),
(-0.999999999999999,-1.73205080756888),(-1.73205080756888,-1))
(1 row)
```

- polygon(npts, circle)

  Description: Circle to npts-point polygon

  Return type: polygon

  Example:

```
gaussdb=# SELECT polygon(12, circle '((0,0),2.0)') AS RESULT;

result

------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------
 ((-2,0),(-1.73205080756888,1),(-1,1.73205080756888),(-1.22464679914735e-16,2),
(1,1.73205080756888),(1.73205080756888,1),(2,2.44929359829471e-16),
(1.73205080756888,-0.999999999999999),(1,-1.73205080756888),(3.67394039744206e-16,-2),
(-0.999999999999999,-1.73205080756888),(-1.73205080756888,-1))
(1 row)
```

- polygon(path)

  Description: Path to polygon

  Return type: polygon

  Example:

```
gaussdb=# SELECT polygon(path '((0,0),(1,1),(2,0))') AS RESULT;
       result
--------------------
 ((0,0),(1,1),(2,0))
(1 row)
```


7.6.11 Network Address Functions and Operators

cidr and inet Operators

The operators **<<**, **<<=**, **>>**, and **>>=** test for subnet inclusion. They consider only the network parts of the two addresses (ignoring any host part) and determine whether one network is identical to or a subnet of the other.

- <

  Description: Is less than

  Example:

```
gaussdb=# SELECT inet '192.168.1.5' < inet '192.168.1.6' AS RESULT;
 result
--------
 t
(1 row)
```

- <=

  Description: Is less than or equals

  Example:

```
gaussdb=# SELECT inet '192.168.1.5' <= inet '192.168.1.5' AS RESULT;
 result
--------
 t
(1 row)
```

- =

  Description: Equals

  Example:

```
gaussdb=# SELECT inet '192.168.1.5' = inet '192.168.1.5' AS RESULT;
 result
--------
 t
(1 row)
```

- >=

  Description: Is greater than or equals

  Example:

```
gaussdb=# SELECT inet '192.168.1.5' >= inet '192.168.1.5' AS RESULT;
 result
--------
 t
(1 row)
```

- >

  Description: Is greater than

  Example:

```
gaussdb=# SELECT inet '192.168.1.5' > inet '192.168.1.4' AS RESULT;
 result
--------
 t
(1 row)
```

- <>

  Description: Does not equal

  Example:

```
gaussdb=# SELECT inet '192.168.1.5' <> inet '192.168.1.4' AS RESULT;
 result
```



```
--------
 t
(1 row)
```

- <<

  Description: Is contained in

  Example:

  ```
  gaussdb=# SELECT inet '192.168.1.5' << inet '192.168.1/24' AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- <<=

  Description: Is contained in or equals

  Example:

  ```
  gaussdb=# SELECT inet '192.168.1/24' <<= inet '192.168.1/24' AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- >>

  Description: Contains

  Example:

  ```
  gaussdb=# SELECT inet '192.168.1/24' >> inet '192.168.1.5' AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- >>=

  Description: Contains or equals

  Example:

  ```
  gaussdb=# SELECT inet '192.168.1/24' >>= inet '192.168.1/24' AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- ~

  Description: Bitwise NOT

  Example:

  ```
  gaussdb=# SELECT ~ inet '192.168.1.6' AS RESULT;
      result
  ---------------
   63.87.254.249
  (1 row)
  ```

- &

  Description: Performs an AND operation on each bit of the two network addresses.

  Example:

  ```
  gaussdb=# SELECT inet '192.168.1.6' & inet '10.0.0.0' AS RESULT;
   result
  ---------
   0.0.0.0
  (1 row)
  ```

- |

Description: Performs an OR operation on each bit of the two network addresses.

Example:

```
gaussdb=# SELECT inet '192.168.1.6' | inet '10.0.0.0' AS RESULT;
   result
-------------
 202.168.1.6
(1 row)
```

- +

  Description: Addition

  Example:

  ```
  gaussdb=# SELECT inet '192.168.1.6' + 25 AS RESULT;
     result
  -------------
   192.168.1.31
  (1 row)
  ```

- -

  Description: Subtraction

  Example:

  ```
  gaussdb=# SELECT inet '192.168.1.43' - 36 AS RESULT;
     result
  -------------
   192.168.1.7
  (1 row)
  ```

- -

  Description: Subtraction

  Example:

  ```
  gaussdb=# SELECT inet '192.168.1.43' - inet '192.168.1.19' AS RESULT;
   result
  --------
      24
  (1 row)
  ```

cidr and inet Functions

The **abbrev**, **host**, and **text** functions are primarily intended to offer alternative display formats.

- abbrev(inet)

  Description: Abbreviated display format as text

  Return type: text

  Example:

  ```
  gaussdb=# SELECT abbrev(inet '10.1.0.0/16') AS RESULT;
     result
  -------------
   10.1.0.0/16
  (1 row)
  ```

- abbrev(cidr)

  Description: Abbreviated display format as text

  Return type: text

  Example:

  ```
  gaussdb=# SELECT abbrev(cidr '10.1.0.0/16') AS RESULT;
   result
  ```



```
---------
 10.1/16
(1 row)
```

- broadcast(inet)

  Description: Broadcast address for networks

  Return type: inet

  Example:

```
gaussdb=# SELECT broadcast('192.168.1.5/24') AS RESULT;
      result
------------------
 192.168.1.255/24
(1 row)
```

- family(inet)

  Description: Extracts family of addresses, **4** for IPv4.

  Return type: int

  Example:

```
gaussdb=# SELECT family('127.0.0.1') AS RESULT;
 result
--------
      4
(1 row)
```

- host(inet)

  Description: Extracts IP addresses as text.

  Return type: text

  Example:

```
gaussdb=# SELECT host('192.168.1.5/24') AS RESULT;
   result
-------------
 192.168.1.5
(1 row)
```

- hostmask(inet)

  Description: Constructs the host mask for a network.

  Return type: inet

  Example:

```
gaussdb=# SELECT hostmask('192.168.23.20/30') AS RESULT;
 result
---------
 0.0.0.3
(1 row)
```

- masklen(inet)

  Description: Extracts subnet mask length.

  Return type: int

  Example:

```
gaussdb=# SELECT masklen('192.168.1.5/24') AS RESULT;
 result
--------
     24
(1 row)
```

- netmask(inet)

  Description: Constructs the subnet mask for a network.

  Return type: inet

Example:

```
gaussdb=# SELECT netmask('192.168.1.5/24') AS RESULT;
   result
---------------
 255.255.255.0
(1 row)
```

- network(inet)

  Description: Extracts the network part of an address.

  Return type: cidr

  Example:

  ```
  gaussdb=# SELECT network('192.168.1.5/24') AS RESULT;
      result
  ----------------
   192.168.1.0/24
  (1 row)
  ```

- set_masklen(inet, int)

  Description: Sets subnet mask length for the **inet** value.

  Return type: inet

  Example:

  ```
  gaussdb=# SELECT set_masklen('192.168.1.5/24', 16) AS RESULT;
      result
  ----------------
   192.168.1.5/16
  (1 row)
  ```

- set_masklen(cidr, int)

  Description: Sets subnet mask length for the **cidr** value.

  Return type: cidr

  Example:

  ```
  gaussdb=# SELECT set_masklen('192.168.1.0/24'::cidr, 16) AS RESULT;
      result
  ----------------
   192.168.0.0/16
  (1 row)
  ```

- text(inet)

  Description: Extracts IP addresses and subnet mask length as text.

  Return type: text

  Example:

  ```
  gaussdb=# SELECT text(inet '192.168.1.5') AS RESULT;
      result
  ----------------
   192.168.1.5/32
  (1 row)
  ```

Any **cidr** value can be cast to **inet** implicitly or explicitly; therefore, the functions shown above as operating on **inet** also work on **cidr** values. An **inet** value can be cast to **cidr**. After the conversion, any bits to the right of the subnet mask are silently zeroed to create a valid **cidr** value. In addition, you can cast a text string to **inet** or **cidr** using normal casting syntax. For example, **inet(expression)** or **colname::cidr**.

macaddr Functions

The function **trunc(macaddr)** returns a MAC address with the last 3 bytes set to zero.

- trunc(macaddr)

  Description: Sets last 3 bytes to zero.

  Return type: macaddr

  Example:

```
gaussdb=# SELECT trunc(macaddr '12:34:56:78:90:ab') AS RESULT;
      result
-------------------
 12:34:56:00:00:00
(1 row)
```

The macaddr type also supports the standard relational operators (such as **>** and **<=**) for lexicographical ordering, and the bitwise arithmetic operators (**~**, **&** and **|**) for NOT, AND and OR.

7.6.12 Text Search Functions and Operators

Text Search Operators

- @@

  Description: Specifies whether the tsvector-typed words match the tsquery-typed words.

  Example:

```
gaussdb=# SELECT to_tsvector('fat cats ate rats') @@ to_tsquery('cat & rat') AS RESULT;
 result
--------
 t
(1 row)
```

- @@@

  Description: Synonym for @@

  Example:

```
gaussdb=# SELECT to_tsvector('fat cats ate rats') @@@ to_tsquery('cat & rat') AS RESULT;
 result
--------
 t
(1 row)
```

- ||

  Description: Connects two tsvector-typed words.

  Example:

```
gaussdb=# SELECT 'a:1 b:2'::tsvector || 'c:1 d:2 b:3'::tsvector AS RESULT;
          result
----------------------------
 'a':1 'b':2,5 'c':3 'd':4
(1 row)
```

- &&

  Description: Performs the AND operation on two tsquery-typed words.

  Example:

```
gaussdb=# SELECT 'fat | rat'::tsquery && 'cat'::tsquery AS RESULT;
          result
```



```
---------------------------
 ( 'fat' | 'rat' ) & 'cat'
(1 row)
```

- ||

  Description: Performs the OR operation on two tsquery-typed words.

  Example:

```
gaussdb=# SELECT 'fat | rat'::tsquery || 'cat'::tsquery AS RESULT;
          result
---------------------------
 ( 'fat' | 'rat' ) | 'cat'
(1 row)
```

- !!

  Description: Not a tsquery-typed word.

  Example:

```
gaussdb=# SELECT !! 'cat'::tsquery AS RESULT;
 result
--------
 !'cat'
(1 row)
```

- @>

  Description: Specifies whether a tsquery-typed word contains another tsquery-typed word.

  Example:

```
gaussdb=# SELECT 'cat'::tsquery @> 'cat & rat'::tsquery AS RESULT;
 result
--------
 f
(1 row)
```

- <@

  Description: Specifies whether a tsquery-typed word is contained in another tsquery-typed word.

  Example:

```
gaussdb=# SELECT 'cat'::tsquery <@ 'cat & rat'::tsquery AS RESULT;
 result
--------
 t
(1 row)
```

In addition to the preceding operators, the ordinary B-tree comparison operators (including = and <) are defined for types tsvector and tsquery.

Text Search Functions

- get_current_ts_config()

  Description: Obtains default text search configurations.

  Return type: regconfig

  Example:

```
gaussdb=# SELECT get_current_ts_config();
 get_current_ts_config
-----------------------
 english
(1 row)
```

- length(tsvector)

  Description: Specifies the number of tsvector-typed words.

Return type: integer

Example:

```
gaussdb=# SELECT length('fat:2,4 cat:3 rat:5A'::tsvector);
 length
--------
      3
(1 row)
```

- numnode(tsquery)

  Description: Specifies the number of tsquery-typed words plus operators.

  Return type: integer

  Example:

  ```
  gaussdb=# SELECT numnode('(fat & rat) | cat'::tsquery);
   numnode
  ---------
        5
  (1 row)
  ```

- plainto_tsquery([config regconfig ,] query text)

  Description: Generates tsquery-typed words without punctuations.

  Return type: tsquery

  Example:

  ```
  gaussdb=# SELECT plainto_tsquery('english', 'The Fat Rats');
   plainto_tsquery
  -----------------
   'fat' & 'rat'
  (1 row)
  ```

- querytree(query tsquery)

  Description: Obtains the indexable part of a tsquery-typed word.

  Return type: text

  Example:

  ```
  gaussdb=# SELECT querytree('foo & ! bar'::tsquery);
   querytree
  -----------
   'foo'
  (1 row)
  ```

- setweight(tsvector, "char")

  Description: Assigns weight to each element of tsvector.

  Return type: tsvector

  Example:

  ```
  gaussdb=# SELECT setweight('fat:2,4 cat:3 rat:5B'::tsvector, 'A');
              setweight
  -------------------------------
   'cat':3A 'fat':2A,4A 'rat':5A
  (1 row)
  ```

- strip(tsvector)

  Description: Removes positions and weights from tsvector.

  Return type: tsvector

  Example:

  ```
  gaussdb=# SELECT strip('fat:2,4 cat:3 rat:5A'::tsvector);
        strip
  -------------------
   'cat' 'fat' 'rat'
  (1 row)
  ```


- to_tsquery([config regconfig ,] query text)

  Description: Normalizes words and converts them to tsquery.

  Return type: tsquery

  Example:

```
gaussdb=# SELECT to_tsquery('english', 'The & Fat & Rats');
  to_tsquery
---------------
 'fat' & 'rat'
(1 row)
```

- to_tsvector([config regconfig ,] document text)

  Description: Reduces document text to tsvector.

  Return type: tsvector

  Example:

```
gaussdb=# SELECT to_tsvector('english', 'The Fat Rats');
   to_tsvector
-----------------
 'fat':2 'rat':3
(1 row)
```

- to_tsvector_for_batch([config regconfig ,] document text)

  Description: Reduces document text to tsvector.

  Return type: tsvector

  Example:

```
gaussdb=# SELECT to_tsvector_for_batch('english', 'The Fat Rats');
   to_tsvector
-----------------
 'fat':2 'rat':3
(1 row)
```

- ts_headline([config regconfig,] document text, query tsquery [, options text])

  Description: Highlights a query match.

  Return type: text

  Example:

```
gaussdb=# SELECT ts_headline('x y z', 'z'::tsquery);
 ts_headline
--------------
 x y <b>z</b>
(1 row)
```

- ts_rank([weights float4[],] vector tsvector, query tsquery [, normalization integer])

  Description: Ranks document for query.

  Return type: float4

  Example:

```
gaussdb=# SELECT ts_rank('hello world'::tsvector, 'world'::tsquery);
 ts_rank
----------
 .0607927
(1 row)
```

- ts_rank_cd([weights float4[],] vector tsvector, query tsquery [, normalization integer])

  Description: Ranks document for query using cover density.

  Return type: float4

Example:

```
gaussdb=# SELECT ts_rank_cd('hello world'::tsvector,'world'::tsquery);
 ts_rank_cd
------------
          0
(1 row)
```

- ts_rewrite(query tsquery, target tsquery, substitute tsquery)

  Description: Replaces tsquery-typed word.

  Return type: tsquery

  Example:

```
gaussdb=# SELECT ts_rewrite('a & b'::tsquery, 'a'::tsquery, 'foo|bar'::tsquery);
       ts_rewrite
-------------------------
 'b' & ( 'foo' | 'bar' )
(1 row)
```

- ts_rewrite(query tsquery, select text)

  Description: Replaces tsquery data in the target with the result of a **SELECT** command.

  Return type: tsquery

  Example:

```
gaussdb=# SELECT ts_rewrite('world'::tsquery, 'select ''world''::tsquery, ''hello''::tsquery');
 ts_rewrite
------------
 'hello'
(1 row)
```

Text Search Debugging Functions

- ts_debug([config regconfig,] document text, OUT alias text, OUT description text, OUT token text, OUT dictionaries regdictionary[], OUT dictionary regdictionary, OUT lexemes text[])

  Description: Tests a configuration.

  Return type: SETOF record

  Example:

```
gaussdb=# SELECT ts_debug('english', 'The Brightest supernovaes');
                                ts_debug
-------------------------------------------------------------------------------
 (asciiword,"Word, all ASCII",The,{english_stem},english_stem,{})
 (blank,"Space symbols"," ",{},,)
 (asciiword,"Word, all ASCII",Brightest,{english_stem},english_stem,{brightest})
 (blank,"Space symbols"," ",{},,)
 (asciiword,"Word, all ASCII",supernovaes,{english_stem},english_stem,{supernova})
(5 rows)
```

- ts_lexize(dict regdictionary, token text)

  Description: Tests a data dictionary.

  Return type: text[]

  Example:

```
gaussdb=# SELECT ts_lexize('english_stem', 'stars');
 ts_lexize
-----------
 {star}
(1 row)
```

- ts_parse(parser_name text, document text, OUT tokid integer, OUT token text)

Description: Tests a parser.

Return type: SETOF record

Example:

```
gaussdb=# SELECT ts_parse('default', 'foo - bar');
 ts_parse
-----------
 (1,foo)
 (12," ")
 (12,"- ")
 (1,bar)
(4 rows)
```

- ts_parse(parser_oid oid, document text, OUT tokid integer, OUT token text)

  Description: Tests a parser.

  Return type: SETOF record

  Example:

```
gaussdb=# SELECT ts_parse(3722, 'foo - bar');
 ts_parse
-----------
 (1,foo)
 (12," ")
 (12,"- ")
 (1,bar)
(4 rows)
```

- ts_token_type(parser_name text, OUT tokid integer, OUT alias text, OUT description text)

  Description: Obtains token types defined by a parser.

  Return type: SETOF record

  Example:

```
gaussdb=# SELECT ts_token_type('default');
                    ts_token_type
-----------------------------------------------------------------
 (1,asciiword,"Word, all ASCII")
 (2,word,"Word, all letters")
 (3,numword,"Word, letters and digits")
 (4,email,"Email address")
 (5,url,URL)
 (6,host,Host)
 (7,sfloat,"Scientific notation")
 (8,version,"Version number")
 (9,hword_numpart,"Hyphenated word part, letters and digits")
 (10,hword_part,"Hyphenated word part, all letters")
 (11,hword_asciipart,"Hyphenated word part, all ASCII")
 (12,blank,"Space symbols")
 (13,tag,"XML tag")
 (14,protocol,"Protocol head")
 (15,numhword,"Hyphenated word, letters and digits")
 (16,asciihword,"Hyphenated word, all ASCII")
 (17,hword,"Hyphenated word, all letters")
 (18,url_path,"URL path")
 (19,file,"File or path name")
 (20,float,"Decimal notation")
 (21,int,"Signed integer")
 (22,uint,"Unsigned integer")
 (23,entity,"XML entity")
(23 rows)
```

- ts_token_type(parser_oid oid, OUT tokid integer, OUT alias text, OUT description text)

  Description: Obtains token types defined by a parser.

  Return type: SETOF record

Example:

```
gaussdb=# SELECT ts_token_type(3722);
                      ts_token_type
------------------------------------------------------------
 (1,asciiword,"Word, all ASCII")
 (2,word,"Word, all letters")
 (3,numword,"Word, letters and digits")
 (4,email,"Email address")
 (5,url,URL)
 (6,host,Host)
 (7,sfloat,"Scientific notation")
 (8,version,"Version number")
 (9,hword_numpart,"Hyphenated word part, letters and digits")
 (10,hword_part,"Hyphenated word part, all letters")
 (11,hword_asciipart,"Hyphenated word part, all ASCII")
 (12,blank,"Space symbols")
 (13,tag,"XML tag")
 (14,protocol,"Protocol head")
 (15,numhword,"Hyphenated word, letters and digits")
 (16,asciihword,"Hyphenated word, all ASCII")
 (17,hword,"Hyphenated word, all letters")
 (18,url_path,"URL path")
 (19,file,"File or path name")
 (20,float,"Decimal notation")
 (21,int,"Signed integer")
 (22,uint,"Unsigned integer")
 (23,entity,"XML entity")
(23 rows)
```

- ts_stat(sqlquery text, [weights text,] OUT word text, OUT ndoc integer, OUT nentry integer)

  Description: Obtains statistics of a tsvector column.

  Return type: SETOF record

  Example:

```
gaussdb=# SELECT ts_stat('select ''hello world''::tsvector');
   ts_stat
-------------
 (world,1,1)
 (hello,1,1)
(2 rows)
```

7.6.13 JSON/JSONB Functions and Operators

For details about the JSON/JSONB data type, see **JSON/JSONB Types**. For details about the operator information, see **Table 7-62** and **Table 7-63**.

Table 7-62 JSON/JSONB common operators

Operators	Left Operand Type	Right Operand Type	Return Type	Description	Example
->	Array-json(b)	int	json(b)	Obtains the **array-json** element. If the index does not exist, **NULL** is returned.	SELECT '[{"a":"foo"},{"b":"bar"},{"c":"baz"}]'::json->2; ?column? ------------- {"c":"baz"} (1 row)

Operators	Left Operand Type	Right Operand Type	Return Type	Description	Example
->	object-json(b)	text	json(b)	Obtains the value by a key. If no record is found, **NULL** is returned.	SELECT '{"a": {"b":"foo"}}'::json->'a'; ?column? ---------- {"b":"foo"} (1 row)
->>	Array-json(b)	int	text	Obtains the JSON array element. If the index does not exist, **NULL** is returned.	SELECT '[1,2,3]'::json->>2; ?column? ---------- 3 (1 row)
->>	object-json(b)	text	text	Obtains the value by a key. If no record is found, **NULL** is returned.	SELECT '{"a":1,"b":2}'::json->>'b'; ?column? ---------- 2 (1 row)
#>	container-json (b)	text[]	json(b)	Obtains the JSON object in the specified path. If the path does not exist, **NULL** is returned.	SELECT '{"a": {"b":{"c": "foo"}}}'::json #>'{a,b}'; ?column? -------------- {"c": "foo"} (1 row)
#>>	container-json (b)	text[]	text	Obtains the JSON object in the specified path. If the path does not exist, **NULL** is returned.	SELECT '{"a":[1,2,3],"b":[4,5,6]}'::json #>>'{a,2}'; ?column? ---------- 3 (1 row)

CAUTION

For the **#>** and **#>>** operators, if no data can be found in the specified path, no error is reported and a **NULL** value is returned.

Table 7-63 Additional JSONB support for operators

Operators	Right Operand Type	Description	Example
@>	jsonb	Specifies whether the top layer of the JSON on the left contains all items of the top layer of the JSON on the right.	'{"a":1, "b":2}'::jsonb @> '{"b":2}'::jsonb
<@	jsonb	Specifies whether all items in the JSON file on the left exist at the top layer of the JSON file on the right.	'{"b":2}'::jsonb <@ '{"a":1, "b":2}'::jsonb
?	text	Specifies whether the string of the key or element exists at the top layer of the JSON value.	'{"a":1, "b":2}'::jsonb ? 'b'
?|	text[]	Specifies whether any of these array strings exists as top-layer keys.	'{"a":1, "b":2, "c":3}'::jsonb ?| array['b', 'c']
?&	text[]	Specifies whether all these array strings exist as top-layer keys.	'["a", "b"]'::jsonb ?& array['a', 'b']
=	jsonb	Determines the size between two **JSONB** files, which is the same as the **jsonb_eq** function.	/
<>	jsonb	Determines the size between two **JSONB** files, which is the same as the **jsonb_ne** function.	/

Operators	Right Operand Type	Description	Example
<	jsonb	Determines the size between two **JSONB** files, which is the same as the **jsonb_lt** function.	/
>	jsonb	Determines the size between two **JSONB** files, which is the same as the **jsonb_gt** function.	/
<=	jsonb	Determines the size between two **JSONB** files, which is the same as the **jsonb_le** function.	/
>=	jsonb	Determines the size between two **JSONB** files, which is the same as the **jsonb_ge** function.	/

Functions Supported by JSON/JSONB

- array_to_json(anyarray [, pretty_bool])

  Description: Returns the array as JSON. It converts a multi-dimensional array into a JSON array. Line feeds will be added between one-dimensional elements if **pretty_bool** is **true**.

  Return type: json

  Example:

```
gaussdb=# SELECT array_to_json('{{1,5},{99,100}}'::int[]);
   array_to_json
   ------------------
   [[1,5],[99,100]]
   (1 row)
```

- row_to_json(record [, pretty_bool])

  Description: Returns the row as JSON. Line feeds will be added between level-1 elements if **pretty_bool** is **true**.

  Return type: json

  Example:

```
gaussdb=# SELECT row_to_json(row(1,'foo'));
    row_to_json
---------------------
```



```
 {"f1":1,"f2":"foo"}
(1 row)
```

- json_array_element(array-json, integer), jsonb_array_element(array-jsonb, integer)

  Description: Same as the operator `->`, which returns the element with the specified subscript in the array.

  Return type: json, jsonb

  Example:

```
gaussdb=# SELECT json_array_element('[1,true,[1,[2,3]],null]',2);
 json_array_element
--------------------
 [1,[2,3]]
(1 row)
```

- json_array_element_text(array-json, integer), jsonb_array_element_text(array-jsonb, integer)

  Description: Same as the operator `->>`, which returns the element with the specified subscript in the array.

  Return type: text, text

  Example:

```
gaussdb=# SELECT json_array_element_text('[1,true,[1,[2,3]],null]',2);
json_array_element_text
------------------------
 [1,[2,3]]
 (1 row)
```

- json_object_field(object-json, text), jsonb_object_field(object-jsonb, text)

  Description: Same as the operator **`->`**, which returns the value of a specified key in an object.

  Return type: json, json

  Example:

```
 gaussdb=# SELECT json_object_field('{"a": {"b":"foo"}}','a');
 json_object_field
 -------------------
 {"b":"foo"}
 (1 row)
```

- json_object_field_text(object-json, text), jsonb_object_field_text(object-jsonb, text)

  Description: Same as the operator **`->`**, which returns the value of a specified key in an object.

  Return type: text, text

  Example:

```
 gaussdb=# SELECT json_object_field_text('{"a": {"b":"foo"}}','a');
 json_object_field_text
 -----------------------
 {"b":"foo"}
 (1 row)
```

- json_extract_path(json, VARIADIC text[]), jsonb_extract_path((jsonb, VARIADIC text[])

  Description: Equivalent to the operator **`#>`** searches for JSON based on the path specified by *$2* and returns the result.

  Return type: json, jsonb

  Example:


```
gaussdb=# SELECT json_extract_path('{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}', 'f4','f6');
 json_extract_path
-------------------
 "stringy"
(1 row)
```

- json_extract_path_op(json, text[]), jsonb_extract_path_op(jsonb, text[])

  Description: Same as the operator `#>` and searches for JSON based on the path specified by *$2* and returns the result.

  Return type: json, jsonb

  Example:

```
gaussdb=# SELECT json_extract_path_op('{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}', ARRAY['f4','f6']);
 json_extract_path_op
----------------------
 "stringy"
(1 row)
```

- json_extract_path_text(json, VARIADIC text[]), jsonb_extract_path_text((jsonb, VARIADIC text[])

  Description: Same as the operator `#>` and searches for JSON based on the path specified by *$2* and returns the result.

  Return type: text, text

  Example:

```
gaussdb=# SELECT json_extract_path_text('{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}', 'f4','f6');
 json_extract_path_text
------------------------
 stringy
(1 row)
```

- json_extract_path_text_op(json, text[]), jsonb_extract_path_text_op(jsonb, text[])

  Description: Same as the operator `#>>` and searches for JSON based on the path specified by *$2* and returns the result.

  Return type: text, text

  Example:

```
gaussdb=# SELECT json_extract_path_text_op('{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}',
ARRAY['f4','f6']);
 json_extract_path_text_op
---------------------------
 stringy
(1 row)
```

- json_array_elements(array-json), jsonb_array_elements(array-jsonb)

  Description: Splits an array. Each element returns a row.

  Return type: json, jsonb

  Example:

```
gaussdb=# SELECT json_array_elements('[1,true,[1,[2,3]],null]');
 json_array_elements
---------------------
 1
 true
 [1,[2,3]]
 null
(4 rows)
```

- json_array_elements_text(array-json), jsonb_array_elements_text(array-jsonb)

  Description: Splits an array. Each element returns a row.

  Return type: text, text

  Example:


```
gaussdb=# SELECT * FROM  json_array_elements_text('[1,true,[1,[2,3]],null]');
    value
-----------
 1
 true
 [1,[2,3]]

 (4 rows)
```

- json_array_length(array-json), jsonb_array_length(array-jsonb)

  Description: Returns the array length.

  Return type: integer

  Example:

```
gaussdb=# SELECT json_array_length('[1,2,3,{"f1":1,"f2":[5,6]},4,null]');
 json_array_length
-------------------
                 6
(1 row)
```

- json_each(object-json), jsonb_each(object-jsonb)

  Description: Splits each key-value pair of an object into one row and two columns.

  Return type: setof(key text, value json), setof(key text, value jsonb)

  Example:

```
gaussdb=# SELECT * FROM  json_each('{"f1":[1,2,3],"f2":{"f3":1},"f4":null}');
  key |  value
-----+----------
 f1  | [1,2,3]
 f2  | {"f3":1}
 f4  | null
(3 rows)
```

- json_each_text(object-json), jsonb_each_text(object-jsonb)

  Description: Splits each key-value pair of an object into one row and two columns.

  Return type: setof(key text, value text), setof(key text, value text)

  Example:

```
gaussdb=# SELECT * FROM  json_each_text('{"f1":[1,2,3],"f2":{"f3":1},"f4":null}');
  key |  value
-----+----------
 f1  | [1,2,3]
 f2  | {"f3":1}
 f4  |
(3 rows)
```

- json_object_keys(object-json), jsonb_object_keys(object-jsonb)

  Description: Returns all keys at the top layer of the object.

  Return type: SETOF text

  Example:

```
gaussdb=# SELECT json_object_keys('{"f1":"abc","f2":{"f3":"a", "f4":"b"}, "f1":"abcd"}');
 json_object_keys
------------------
 f1
 f2
 f1
(3 rows)
```

- JSONB deduplication operations:

```
gaussdb=# SELECT jsonb_object_keys('{"f1":"abc","f2":{"f3":"a", "f4":"b"}, "f1":"abcd"}');
 jsonb_object_keys
-------------------
 f1
```



```
 f2
(2 rows)
```

- json_populate_record(anyelement, object-json [, bool]), jsonb_populate_record(anyelement, object-jsonb [, bool])

  Description: *$1* must be a compound parameter. Each key-value in the **object-json** file is split. The key is used as the column name to match the column name in *$1* and fill in the *$1* format.

  Return type: anyelement, anyelement

  Example:

```
gaussdb=# CREATE TYPE jpop AS (a text, b int, c bool);
CREATE TYPE
gaussdb=# SELECT * FROM json_populate_record(null::jpop,'{"a":"blurfl","x":43.2}');
   a    | b | c
--------+---+---
 blurfl |   |
(1 row)

gaussdb=# SELECT * FROM json_populate_record((1,1,null)::jpop,'{"a":"blurfl","x":43.2}');
   a    | b | c
--------+---+---
 blurfl | 1 |
(1 row)
gaussdb=# DROP TYPE jpop;
DROP TYPE
```

- json_populate_record_set(anyelement, array-json [, bool]), jsonb_populate_record_set(anyelement, array-jsonb [, bool])

  Description: Performs the preceding operations on each element in the *$2* array by referring to the **json_populate_record** and **jsonb_populate_record** functions. Therefore, each element in the *$2* array must be of the **object-json** type.

  Return type: setof anyelement, setof anyelement

  Example:

```
gaussdb=# CREATE TYPE jpop AS (a text, b int, c bool);
CREATE TYPE
gaussdb=# SELECT * FROM json_populate_recordset(null::jpop, '[{"a":1,"b":2},{"a":3,"b":4}]');
 a | b | c
---+---+---
 1 | 2 |
 3 | 4 |
(2 rows)
gaussdb=# DROP TYPE jpop;
DROP TYPE
```

- json_typeof(json), jsonb_typeof(jsonb)

  Description: Checks the JSON type.

  Return type: text, text

  Example:

```
gaussdb=# SELECT value, json_typeof(value)  FROM (values (json '123.4'), (json '"foo"'), (json 'true'), (json 'null'), (json '[1, 2, 3]'), (json '{"x":"foo", "y":123}'), (NULL::json))  AS data(value);
          value          | json_typeof
-------------------------+-------------
 123.4                   | number
 "foo"                   | string
 true                    | boolean
 null                    | null
 [1, 2, 3]               | array
 {"x":"foo", "y":123} | object
                         |
(7 rows)
```


- json_build_array([VARIADIC "any"])

  Description: Constructs a JSON array from a variable parameter list.

  Return type: array-json

  Example:

```
gaussdb=# SELECT json_build_array('a',1,'b',1.2,'c',true,'d',null,'e',json '{"x": 3, "y": [1,2,3]}','');
                          json_build_array
------------------------------------------------------------------------------
 ["a", 1, "b", 1.2, "c", true, "d", null, "e", {"x": 3, "y": [1,2,3]}, null]
(1 row)
```

- json_build_object([VARIADIC "any"])

  Description: Constructs a JSON object from a variable parameter list. The number of input parameters must be an even number. Every two input parameters form a key-value pair. Note that the value of a key cannot be null.

  Return type: object-json

  Example:

```
gaussdb=# SELECT json_build_object(1,2);
  json_build_object
 -------------------
  {"1" : 2}
 (1 row)
```

- jsonb_build_object([VARIADIC "any"])

  Description: Constructs a JSONB object from a variable parameter list. The number of input parameters must be an even number. Every two input parameters form a key-value pair. Note that the key value cannot be **NULL**.

  Return type: object-jsonb

  NOTICE

  - When an element in the variable parameter list contains an empty string (''), if the SQL compatibility mode of the database is A, the return result of the corresponding element is **NULL**; if the SQL compatibility mode of the database is PG, the return result of the corresponding element is an empty string. This is because empty strings are treated as **NULL** in A compatibility mode.
  - When an element in the variable parameter list is of the DATE type, if the SQL compatibility mode of the database is A, the return result of the corresponding element contains hour, minute, and second; if the SQL compatibility mode of the database is PG, the return result of the corresponding element does not contain hour, minute, and second. This is because DATE will be replaced by TIMESTAMP(0) WITHOUT TIME ZONE in A compatibility mode.

  Example:

```
gaussdb=# SELECT jsonb_build_object('a',1,'b',1.2,'c',true,'d',null,'e',json '{"x": 3, "y": [1,2,3]}');
                          jsonb_build_object
------------------------------------------------------------------------------
 {"a": 1, "b": 1.2, "c": true, "d": null, "e": {"x": 3, "y": [1, 2, 3]}}
(1 row)

gaussdb=# SELECT jsonb_build_object();
 jsonb_build_object
--------------------
```



```
 {}
(1 row)
```

- json_to_record(object-json, bool), json_to_record(object-json)

  Description: Like all functions that return **record**, the caller must explicitly define the structure of the record with an AS clause. The key-value pair of **object-json** is split and reassembled. The key is used as a column name to match and fill in the structure of the record explicitly specified by the AS clause.. The **bool** input parameter of overload 1 specifies whether object nesting is allowed. The value **true** indicates yes and the value **false** indicates no. If there is only one input parameter, object nesting is allowed by default. If there is only one input parameter, the nested JSON object can be converted to the row type and arrays in square brackets can be parsed. Dates of the time type are equivalent to timestamp(0) in A compatibility mode. Compared with that in the PG compatibility mode, the value also displays the 00:00:00 (hour, minute, and second) character string.

  Return type: record

  Example:

```
gaussdb=# SELECT * FROM json_to_record('{"a":1,"b":"foo","c":"bar"}',true) AS x(a int, b text, d text);
  a |  b  | d
 ---+-----+---
  1 | foo |
 (1 row)
gaussdb=# SELECT * FROM json_to_record('{"a": {"x": 1, "y": 2},"b":"foo","c":[1, 2]}') AS x(a json, b
text, c int[]);
         a         | b |   c
 ------------------+-----+-------
  {"x": 1, "y": 2} | foo | {1,2}
 (1 row)
```

- json_to_recordset(array-json, bool)

  Description: Executes the preceding function on each element in the array by referring to the json_to_record function. Therefore, each element in the array must be object-json. For **bool**, refer to the json_to_record function. **true** indicates that nested objects can be parsed, and **false** indicates that nested objects cannot be parsed, that is, whether the value of an element in a JSON object can be a JSON object.

  Return type: SETOF record

  Example:

```
gaussdb=# SELECT * FROM json_to_recordset('[{"a":1,"b":"foo","d":false},
{"a":2,"b":"bar","c":true}]',false) AS x(a int, b text, c boolean);
 a |  b  | c
---+-----+---
 1 | foo |
 2 | bar | t
(2 rows)
```

- json_object(text[]), json_object(text[], text[])

  Description: Constructs an **object-json** from a text array. This is an overloaded function. When the input parameter is a text array, the array length must be an even number, and members are considered as alternate key-value pairs. When two text arrays are used, the first array is considered as a key, and the second array a value. The lengths of the two arrays must be the same. Note that the value of a key cannot be null.

  Return type: object-json

  Example:

```
gaussdb=# SELECT json_object('{a,1,b,2,3,NULL,"d e f","a b c"}');
                json_object
```



```
-------------------------------------------------------
 {"a" : "1", "b" : "2", "3" : null, "d e f" : "a b c"}
(1 row)
gaussdb=# SELECT json_object('{a,b,"a b c"}', '{a,1,1}');
              json_object
---------------------------------------
 {"a" : "a", "b" : "1", "a b c" : "1"}
(1 row)
```

- json_object([VARIADIC "any"])

  Description: Constructs a JSON object from a variable parameter list. The number of input parameters must be an even number. Every two input parameters form a key-value pair. If the key-value pair is null, an error is reported. If the parameter is in the odd format, an error is reported.

  Parameter: variable parameter list. The input parameter is a combination of one or more key-value pairs.

  Return type: json

  Example:

```
gaussdb=# SELECT json_object('d',2,'c','name','b',true,'a',2,'a',NULL,'d',1);
              json_object
-----------------------------------------
 {"a": 2, "b": true, "c": "name", "d": 2}
(1 row)

gaussdb=# SELECT json_object('d',2,true,'name','b',true,'a',2,'aa', current_timestamp);
                                 json_object
-----------------------------------------------------------------------------
 {"1": "name", "a": 2, "b": true, "d": 2, "aa": "2023-08-12 11:28:13.385958"}
(1 row)
```

  NOTE

  This function takes effect when **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'** in the B-compatible database. When this function takes effect, the original json_object behavior is replaced.

- json_append/json_array_append(json, [VARIADIC "any"])

  Description: Constructs several json_path-value pairs from a variable parameter list, adds a value to the path specified by **json**, and returns the modified JSON value. json_append is the same as json_array_append. If any parameter is null, **null** is returned. If the JSON format is incorrect, **json_path** is an invalid path expression, or **json_path** contains * or **, an error is reported. If the variable parameter list contains null values and has format error, the exceptions are processed based on the exception sequence. If a null value exists first, **null** is returned first. If a format error occurs first, an error is reported first.

  Parameter: The first input parameter must be in JSON format, and the second input parameter is a variable parameter list. The json_path-value pair is constructed from the variable parameter list. For details, see **Table 7-64**.

Table 7-64 Parameters

Parameter	Type	Description	Value Range
json	json	JSON value to be modified.	The value must be in JSON format.

Parameter	Type	Description	Value Range
[VARIADIC "any"]	Variable any array	A variable parameter list, including variable json_path-value pairs.	**json_path** must be a valid path expression, and **value** can be any value.

Return type: json

Example:

```
gaussdb=# SELECT json_array_append('[1, [2, 3]]', '$[1]', 4, '$[0]', false, '$[0]', null, '$[0]',
current_timestamp);
                      json_array_append
---------------------------------------------------------------
 [[1, false, null, "2023-08-12 14:27:16.142355+08"], [2, 3, 4]]
(1 row)
```

- json_array([VARIADIC "any"])

  Description: Constructs an array from a variable parameter list and returns a JSON array. If the function does not have any parameters, an empty JSON array is returned.

  Parameter: The input parameter is a variable parameter list. The values in the list can be of any type.

  Return type: json

  Example:

  ```
  -- If no input parameter is entered, an empty JSON array is returned.
  gaussdb=# SELECT json_array();
   json_array
  ------------
   []
  (1 row)

  -- The input parameter can be of any type.
  gaussdb=# SELECT json_array(TRUE, FALSE, NULL, 114, 'text', current_timestamp);
                          json_array
  ------------------------------------------------------------------
   [true, false, null, 114, "text", "2023-08-12 15:17:34.979294+08"]
  (1 row)
  ```

- json_array_insert(json, [VARIADIC "any"])

  Description: Constructs one or more json_path-value pairs from a variable parameter list, inserts a value into the array path specified by the **json_path** in **json**, and returns a new JSON value. If there is already an existing value on the specified path, the value is inserted on the path and the existing value is moved backward. If **json** or **json_path** is null, **null** is returned. If the JSON format is incorrect, **json_path** is an invalid path expression, or **json_path** contains * or **, an error is reported. If the variable parameter list contains null values and has format error, the exceptions are processed based on the exception sequence. If a null value exists first, **null** is returned first. If a format error occurs first, an error is reported first.

  Parameters: For details, see **Table 7-65**.

Table 7-65 Parameters

Parameter	Type	Description	Value Range
json	json	JSON value to be modified.	The value must be in JSON format.
[VARIADIC "any"]	Variable any array	Variable parameter list, containing one or more json_path-value pairs.	**json_path** must be a valid path expression, and **value** can be any value.

Return type: json

Example:

```
-- Example of containing one json_path-value pair
gaussdb=# SELECT json_array_insert('[1, [2, 3]]', '$[1]', 4);
 json_array_insert
-------------------
 [1, 4, [2, 3]]
(1 row)

-- Example of containing multiple json_path-value pairs
gaussdb=# SELECT json_array_insert('{"x": 1, "y": [1, 2]}', '$.y[0]', NULL, '$.y[0]', 123, '$.y[3]',
current_timestamp);
                        json_array_insert
------------------------------------------------------------------
 {"x": 1, "y": [123, null, 1, "2023-08-14 14:54:12.85087+08", 2]}
(1 row)
```

- json_contains(target_json, candidate_json[, json_path])

  Description: The input parameters are two JSON objects and an optional path specified by **json_path**. If **json_path** is not specified, the system checks whether **target_json** contains **candidate_json**. If **json_path** is specified, the system checks whether **candidate_json** is contained in the JSON path specified by **json_path** in **target_json**. If **json** or **json_path** is null, **null** is returned. If the JSON format is incorrect, **json_path** is an invalid path expression, or **json_path** contains * or **, an error is reported. If the variable parameter list contains null values and has format error, the exceptions are processed based on the exception sequence. If a null value exists first, **null** is returned first. If a format error occurs first, an error is reported first.

  Parameters: For details, see **Table 7-66**.

Table 7-66 Parameters

Parameter	Type	Description	Value Range
target_json	json	Target JSON object, which is used to check whether **candidate_json** is contained in the JSON object.	The value must be in JSON format.

Parameter	Type	Description	Value Range
candidate_json	json	JSON subobject, which is used to check whether it is contained in **target_json**.	The value must be in JSON format.
json_path	text	Specified JSON path. This parameter is optional. If this parameter is specified, the path specified by **json_path** in **target_json** is used to determine the inclusion relationship.	**json_path** must be a valid path expression.

Return type: bigint. If there is an inclusion relationship, **1** is returned. If there is no inclusion relationship, **0** is returned.

Example:

```
-- No path is specified.
gaussdb=# SELECT json_contains('[1, 2, {"x": 3}]', '{"x":3}');
 json_contains
---------------
             1
(1 row)

-- A path is specified.
gaussdb=# SELECT json_contains('[1, 2, {"x": 3},[1,2,3,4]]', '2','$[1]');
 json_contains
---------------
             1
(1 row)

gaussdb=# SELECT json_contains('[1, 2, {"x": 3},[1,2,3,4]]', '1','$[1]');
 json_contains
---------------
             0
(1 row)
```

- json_contains_path(json, mode_str, [VARIADIC text])

  Description: Checks whether a JSON object has a value on the specified path. There can be multiple paths. The first input parameter is a JSON object, and the second input parameter can be **one** or **all**, specifying whether to check all paths. The third input parameter is a variable parameter list. All JSON paths are constructed from the variable parameter list. According to the mode, if a path value exists, **1** is returned. If no path value exists, **0** is returned. If **json** or **json_path** is null, **null** is returned. If the JSON format is incorrect, or **json_path** is an invalid path expression, an error is reported. If the variable parameter list contains null values and has format error, the exceptions are processed based on the exception sequence. If a null value exists first, **null** is returned first. If a format error occurs first, an error is reported first.

  Parameters: For details, see **Table 7-67**.

Table 7-67 Parameters

Parameter	Type	Description	Value Range
json	json	JSON object to be transferred.	The value must be in JSON format.
mode_str	text	The value can be **one** or **all**. If the value is **one** and one path exists, **1** is returned; otherwise, **0** is returned. If the value is **all** and all paths exist, **1** is returned; otherwise, **0** is returned.	The value must be **one** or **all**, which is case insensitive.
[VARIADIC text]	Variable text array	Variable parameter list from which all JSON paths are constructed.	**json_path** must be a valid path expression.

Return type: bigint

Example:

```
-- In the all scenario, all paths must exist.
gaussdb=# SELECT json_contains_path('[1, 2, {"x": 3}]', 'all', '$[0]', '$[1]', '$[2]');
 json_contains_path
--------------------
                  1
(1 row)

gaussdb=# SELECT json_contains_path('[1, 2, {"x": 3}]', 'all', '$[0]', '$[1]', '$[6]');
 json_contains_path
--------------------
                  0
(1 row)

-- In the one scenario, at least one path must exist.
gaussdb=# SELECT json_contains_path('[1, 2, {"x": 3}]', 'one', '$[0]', '$[1]', '$[5]');
 json_contains_path
--------------------
                  1
(1 row)
```

- json_depth(json)

  Description: The input parameter is a JSON object. This function is used to return the maximum depth of the JSON object. If the input parameter is null, **null** is returned.

  Parameter: The input parameter is a JSON string whose depth needs to be queried. If the input parameter is not in valid JSON format, an error is reported.

  Return type: int

  Example:

```
-- The depth of an empty array is 1.
gaussdb=# SELECT json_depth('[]');
 json_depth
------------
          1
```



```
(1 row)

gaussdb=# SELECT json_depth('{"s":1, "x":2,"y":[1]}');
 json_depth
------------
          3
(1 row)
```

- json_extract(json, [VARIADIC text])

  Description: The input parameters are a JSON object and several JSON paths. The JSON paths are constructed from the variable parameter list. This function extracts data of the specified JSON path from **json**, combines the data into a JSON array, and returns the JSON array. If **json** or **json_path** is null, **null** is returned. If the JSON format is incorrect, or **json_path** is an invalid path expression, an error is reported. If the variable parameter list contains null values and has format error, the exceptions are processed based on the exception sequence. If a null value exists first, **null** is returned first. If a format error occurs first, an error is reported first.

  Parameters: For details, see **Table 7-68**.

  Table 7-68 Parameters

Parameter	Type	Description	Value Range
json	json	JSON value from which paths are to be extracted.	The value must be in JSON format.
[VARIADIC text]	Variable text array	Variable parameter list, containing one or more JSON paths.	**json_path** must be a valid path expression.

  Return type: json

  Example:

```
-- One path is extracted.
gaussdb=# SELECT json_extract('[1, 2, {"x": 3}]', '$[2]');
 json_extract
--------------
 {"x": 3}
(1 row)

-- Multiple paths are extracted.
gaussdb=# SELECT json_extract('["a", ["b", "c"], "d"]', '$[1]', '$[2]', '$[3]');
   json_extract
-------------------
 [["b", "c"], "d"]
(1 row)
```

- json_insert(json, [VARIADIC any])

  Description: The input parameter is a JSON object, multiple JSON paths, and values to be inserted. JSON paths and values must be paired. This function inserts **value** at the position specified by **json_path** in **json**. This function can only insert data to a position where no path exists. If the specified path already exists in **json**, no data will be inserted. If **json** or **json_path** is null, **null** is returned. In addition, **json_path** cannot contain wildcard characters * or **. Otherwise, an error is reported. If the variable parameter list contains

null values and has format error, the exceptions are processed based on the exception sequence. If a null value exists first, **null** is returned first. If a format error occurs first, an error is reported first.

Parameter: For details, see **Table 8 Parameters**.

Table 7-69 Parameters

Parameter	Type	Description	Value Range
json	json	JSON object to which a value is to be inserted.	The value must be in JSON format.
[VARIADIC any]	Variable any array	Variable parameter list, containing one or more json_path-value pairs.	**json_path** must be a valid path expression, and **value** can be any value.

Return type: json

Example:

```
gaussdb=# SELECT json_insert('[1, [2, 3], {"a": [4, 5]}]', '$[3]', 2);
         json_insert
-----------------------------
 [1, [2, 3], {"a": [4, 5]}, 2]
(1 row)

gaussdb=# SELECT json_insert('[1, [2, 3], {"a": [4, 5]}]', '$[10]', 10,'$[5]', 5);
           json_insert
---------------------------------
 [1, [2, 3], {"a": [4, 5]}, 10, 5]
(1 row)
```

- json_keys(json[, json_path])

  Description: The input parameters are a JSON object and an optional JSON path. If no JSON path is transferred, this function returns a JSON array of member key values of the top-layer object in the JSON object. If a JSON path is transferred, this function returns the JSON array of the top-layer member key values in the JSON object corresponding to the path. If **json** or **json_path** is null, **null** is returned. If the JSON format is incorrect, **json_path** is an invalid path expression, or **json_path** contains * or **, an error is reported.

  Parameter: For details, see **Table 9 Parameters**.

  Table 7-70 Parameters

Parameter	Type	Description	Value Range
json	json	A JSON value.	The value must be in JSON format.
json_path	text	A JSON path, which is optional.	**json_path** must be a valid path expression.

Return type: json

Example:

```
gaussdb=# SELECT json_keys('{"x": 1, "y": 2, "z": 3}');
    json_keys
-----------------
 ["x", "y", "z"]
(1 row)

gaussdb=# SELECT json_keys('[1,2,3,{"name":"Tom"}]','$[3]');
 json_keys
-----------
 ["name"]
(1 row)
```

- json_length(json[, json_path])

  Description: The input parameters are a JSON object and an optional JSON path. If no JSON path is transferred, this function returns the length of the transferred JSON object. If a JSON path is transferred, this function returns the length of the JSON object corresponding to the path. If **json** or **json_path** is null, **null** is returned. If the JSON format is incorrect, **json_path** is an invalid path expression, or **json_path** contains * or **, an error is reported.

  Parameters: For details, see **Table 7-71**.

  Table 7-71 Parameters

Parameter	Type	Description	Value Range
json	json	A JSON value.	The value must be in JSON format.
json_path	text	A JSON path, which is optional.	**json_path** must be a valid path expression.

  Return type: int

  Example:

```
gaussdb=# SELECT json_length('[1,2,3,4,5]');
 json_length
-------------
           5
(1 row)

gaussdb=# SELECT json_length('{"name":"Tom", "age":24, "like":"football"}');
 json_length
-------------
           3
(1 row)
```

- json_merge([VARIADIC any])

  Description: The input parameters are of the JSON type and the number of input parameters is greater than or equal to 2. The JSON objects are constructed from the variable parameter list. This function combines all input JSON parameters and returns the combination result. If the input parameter is null, **null** is returned. If the variable parameter list contains null values and has format error, the exceptions are processed based on the exception sequence. If a null value exists first, **null** is returned first. If a format error occurs first, an error is reported first.

Parameter: a variable parameter list. Multiple JSON objects are constructed from this list.

Return type: json

Example:

```
gaussdb=# SELECT json_merge('[1, 2]','[2]');
 json_merge
------------
 [1, 2, 2]
(1 row)

gaussdb=# SELECT json_merge('{"b":"2"}','{"a":"1"}','[1,2]');
          json_merge
-------------------------------
 [{"a": "1", "b": "2"}, 1, 2]
(1 row)
```

- json_quote(text)

  Description: The input parameter is of the text type. This function uses double quotation marks ("") to enclose the input parameter into a JSON string and returns the string.

  Parameter: The input parameter is the character string to be enclosed.

  Return type: json

  Example:

  ```
  gaussdb=# SELECT json_quote('gauss');
   json_quote
  ------------
   "gauss"
  (1 row)
  ```

- json_unquote(json)

  Description: The input parameter is a JSON object. This function cancels the quotation marks of the input parameter and returns the character string.

  Return type: object-json

  Return type: text

  Example:

  ```
  gaussdb=# SELECT json_unquote('"gauss"');
   json_unquote
  --------------
   gauss
  (1 row)
  ```

- json_remove(json, [VARIADIC text])

  Description: The input parameters are a JSON object and several JSON paths to be deleted. These JSON paths are constructed from the variable parameter list. This function deletes the value of **json_path** in json and returns the JSON object after deletion. If **json** or **json_path** is null, **null** is returned. If the JSON format is incorrect, **json_path** is an invalid path expression, or **json_path** contains * or **, an error is reported. If the variable parameter list contains null values and has format error, the exceptions are processed based on the exception sequence. If a null value exists first, **null** is returned first. If a format error occurs first, an error is reported first.

  Parameters: For details, see **Table 7-72**.

Table 7-72 Parameters

Parameter	Type	Description	Value Range
json	json	A JSON value.	The value must be in JSON format.
[VARIADIC text]	Variable text array	Variable parameter list, containing one or more JSON paths.	**json_path** must be a valid path expression.

Return type: json

Example:

```
gaussdb=# SELECT json_remove('[0, 1, 2, [3, 4]]', '$[0]', '$[2]');
 json_remove
-------------
 [1, 2]
(1 row)

gaussdb=# SELECT json_remove('[0, 1, 2, [3, 4]]', '$[0]', '$[0]','$[0]');
 json_remove
-------------
 [[3, 4]]
(1 row)
```

- json_replace(json, [VARIADIC any])

  Description: The input parameter is a JSON object, multiple JSON paths, and values to be replaced. The JSON paths and values must be in pairs. This function replaces data in a path specified by **json_path** in **json** with a specified value and returns the modified JSON object. If **json** or **json_path** is null, **null** is returned. If the JSON format is incorrect, **json_path** is an invalid path expression, or **json_path** contains * or **, an error is reported. If the variable parameter list contains null values and has format error, the exceptions are processed based on the exception sequence. If a null value exists first, **null** is returned first. If a format error occurs first, an error is reported first.

  Parameters: For details, see **Table 7-73**.

  Table 7-73 Parameters

Parameter	Type	Description	Value Range
json	json	A JSON value.	The value must be in JSON format.
[VARIADIC any]	Variable any input parameter	Variable parameter list, containing one or more json_path-value pairs.	**json_path** must be a valid path expression, and **value** can be any value.

  Return type: json

Example:

```
gaussdb=# SELECT json_replace('{"x": 1}', '$.x', 'true');
 json_replace
---------------
 {"x": "true"}
(1 row)

gaussdb=# SELECT json_replace('{"x": 1}', '$.x', true, '$.x', 123, '$.x', 'asd', '$.x', null);
 json_replace
--------------
 {"x": null}
(1 row)
```

- json_search(json, mode_str, search_str, escape_char, json_path)

  Description: Returns the path of a given string in **json**. It returns a path string or an array of multiple paths. If **json**, **search_str**, or **json_path** is null, **null** is returned. If the JSON format is incorrect or **json_path** is not a valid path expression, an error is reported. If the variable parameter list contains null values and has format error, the exceptions are processed based on the exception sequence. If a null value exists first, **null** is returned first. If a format error occurs first, an error is reported first.

  Parameter: For details, see **Table 13 Parameters**.

  Table 7-74 Parameters

Parameter	Type	Description	Value Range
json	json	Target JSON object.	The value must be in JSON format.
mode_str	text	The value can be **one** or **all**. If the value is **one**, only the first matched path is obtained. If the value is **all**, all matched paths are obtained.	The value must be **one** or **all**, which is case insensitive.
search_str	text	Character string to be searched for. In normal cases, the entire character string is matched. However, the wildcard % can be used to match any number of characters, and _ can be used to match any character.	-

Parameter	Type	Description	Value Range
escape_char	text	If **search_str** contains a wildcard, this parameter is defined as an escape character for the wildcard. This parameter is optional. The default value is **'\'**. If the character is added before the wildcard, the wildcard is not considered as a wildcard.	The value can be any single character.
json_path	text	If **json_path** is specified, the search is performed in the path. This parameter is optional.	**json_path** must be a valid path expression.

Return type: text

Example:

```
-- In the all mode:
gaussdb=# SELECT json_search('{"a":"abc","b":"abc"}','all','abc');
  json_search
----------------
 ["$.a", "$.b"]
(1 row)

-- In the one mode:
gaussdb=# SELECT json_search('{"a":"abc","b":"abc"}','one','abc');
 json_search
-------------
 "$.a"
(1 row)

-- The default escape character is used.
gaussdb=# SELECT json_search('{"a":"abc","b":"a%c"}','one','a\%c');
 json_search
-------------
 "$.b"
(1 row)
```

- json_set(json, [VARIADIC any])

  Description: The input parameter is a JSON object, multiple JSON paths, and values to be set. The JSON paths and values must be in pairs. This function uses the specified value to update the data in the path specified by **json_path** in **json** and returns the modified JSON object. If the specified path does not have any value, the value is inserted into the corresponding path. If **json** or **json_path** is null, **null** is returned. If the JSON format is incorrect, **json_path** is an invalid path expression, or **json_path** contains * or **, an error is reported. If the variable parameter list contains null values and has format error, the exceptions are processed based on the exception sequence. If a null value exists first, **null** is returned first. If a format error occurs first, an error is reported first.

  Parameters: For details, see **Table 7-75**.

Table 7-75 Parameters

Parameter	Type	Description	Value Range
json	json	A JSON value.	The value must be in JSON format.
[VARIADIC any]	Variable any array	Variable parameter list, containing one or more json_path-value pairs.	**json_path** must be a valid path expression, and **value** can be any value.

Return type: json

Example:

```
gaussdb=# SELECT json_set('{"s":3}','$.s','d');
  json_set
------------
 {"s": "d"}
(1 row)

gaussdb=# SELECT json_set('{"s":3}','$.a','d','$.a','1');
      json_set
--------------------
 {"a": "1", "s": 3}
(1 row)
```

- json_type(json)

  Description: The input parameter is a JSON object. This function returns a string, which represents the type of the given JSON value. If the input parameter is null, **null** is returned.

  Parameter: a JSON value. For details, see **Table 7-76**.

  Return type: text

Table 7-76 Return value of json_type

Input Parameter Type	Return Value
JSON value of the array type	ARRAY
JSON value of the object type	OBJECT
JSON value of the character string type	STRING
JSON value of the numeric type	NUMBER
JSON value of the Boolean type	BOOLEAN
NULL	NULL

Example:

```
gaussdb=# SELECT json_type('{"w":{"2":3},"2":4}');
 json_type
-----------
```



```
 OBJECT
(1 row)

gaussdb=# SELECT json_type('[1,2,2,3,3,4,4,4,4,4,4,4,4]');
 json_type
-----------
 ARRAY
(1 row)
```

- json_valid(json)

  Description: Returns **0** or **1** to indicate whether the given parameter is a valid JSON object. If the input parameter is null, **null** is returned.

  Return type: object-json

  Return type: bigint

  Example:

```
gaussdb=# SELECT json_valid('{"name":"Tom"}');
 json_valid
------------
          1
(1 row)


gaussdb=# SELECT json_valid('[1,23,4,5,5]');
 json_valid
------------
          1
(1 row)

gaussdb=# SELECT json_valid('[1,23,4,5,5]}');
 json_valid
------------
          0
(1 row)
```

- json_agg(any)

  Description: Aggregates values into a JSON array.

  Return type: array-json

  Example:

```
gaussdb=# CREATE TABLE classes(name varchar, score int);
CREATE TABLE
gaussdb=# INSERT INTO classes VALUES('A',2);
INSERT 0 1
gaussdb=# INSERT INTO classes VALUES('A',3);
INSERT 0 1
gaussdb=# INSERT INTO classes VALUES('D',5);
INSERT 0 1
gaussdb=# INSERT INTO classes VALUES('D',null);
INSERT 0 1
gaussdb=# SELECT * FROM classes;
 name | score
------+-------
 A    |     2
 A    |     3
 D    |     5
 D    |
(4 rows)

gaussdb=# SELECT name, json_agg(score) score FROM classes GROUP BY name ORDER BY name;
 name |   score
------+-----------
 A    | [2, 3]
 D    | [5, null]
(2 rows)

gaussdb=# DROP TABLE classes;
DROP TABLE
```


- json_object_agg(any, any)

  Description: Aggregates values into a JSON object.

  Return type: object-json

  Example:

```
gaussdb=# CREATE TABLE classes(name varchar, score int);
CREATE TABLE
gaussdb=# INSERT INTO classes VALUES('A',2);
INSERT 0 1
gaussdb=# INSERT INTO classes VALUES('A',3);
INSERT 0 1
gaussdb=# INSERT INTO classes VALUES('D',5);
INSERT 0 1
gaussdb=# INSERT INTO classes VALUES('D',null);
INSERT 0 1
gaussdb=# SELECT * FROM classes;
 name | score
------+-------
 A    |     2
 A    |     3
 D    |     5
 D    |
(4 rows)

gaussdb=# SELECT json_object_agg(name, score) FROM classes GROUP BY name ORDER BY name;
     json_object_agg
-------------------------
 { "A" : 2, "A" : 3 }
 { "D" : 5, "D" : null }
(2 rows)

gaussdb=# DROP TABLE classes;
DROP TABLE
```

- jsonb_contained(jsonb, jsonb)

  Description: Same as the operator **`<@`**, determines whether all elements in *$1* exist at the top layer of *$2*.

  Return type: Boolean

  Example:

```
 gaussdb=# SELECT jsonb_contained('[1,2,3]', '[1,2,3,4]');
  jsonb_contained
 -----------------
  t
 (1 row)
```

- jsonb_contains(jsonb, jsonb)

  Description: Same as the operator **`@>`**, checks whether all top-layer elements in *$1* are contained in *$2*.

  Return type: Boolean

  Example:

```
 gaussdb=# SELECT jsonb_contains('[1,2,3,4]', '[1,2,3]');
  jsonb_contains
 ----------------
  t
 (1 row)
```

- jsonb_exists(jsonb, text)

  Description: Same as the operator **`?`**, determines whether all elements in the string array *$2* exist at the top layer of *$1* in the form of **key\elem\scalar**.

  Return type: Boolean

  Example:


```
gaussdb=# SELECT jsonb_exists('["1",2,3]', '1');
 jsonb_exists
--------------
 t
(1 row)
```

- jsonb_exists_all(jsonb, text[])

  Description: Same as the operator **`?&`**, checks whether all elements in the string array *$2* exist at the top layer of *$1* in the form of **key\elem\scalar**.

  Return type: Boolean

  Example:

```
gaussdb=# SELECT jsonb_exists_all('["1","2",3]', '{1, 2}');
 jsonb_exists_all
------------------
 t
(1 row)
```

- jsonb_exists_any(jsonb, text[])

  Description: Same as the operator **`?|`**, checks whether all elements in the string array *$2* exist at the top layer of *$1* in the form of **key\elem\scalar**.

  Return type: Boolean

  Example:

```
gaussdb=# SELECT jsonb_exists_any('["1","2",3]', '{1, 2, 4}');
 jsonb_exists_any
------------------
 t
(1 row)
```

- jsonb_cmp(jsonb, jsonb)

  Description: Compares values. A positive value indicates greater than, a negative value indicates less than, and **0** indicates equal.

  Return type: integer

  Example:

```
gaussdb=# SELECT jsonb_cmp('["a", "b"]', '{"a":1, "b":2}');
  jsonb_cmp
-----------
         -1
(1 row)
```

- jsonb_eq(jsonb, jsonb)

  Description: Same as the operator **`=`**, compares two values.

  Return type: Boolean

  Example:

```
gaussdb=# SELECT jsonb_eq('["a", "b"]', '{"a":1, "b":2}');
 jsonb_eq
----------
 f
(1 row)
```

- jsonb_ne(jsonb, jsonb)

  Description: Same as the operator **`<>`**, compares two values.

  Return type: Boolean

  Example:

```
gaussdb=# SELECT jsonb_ne('["a", "b"]', '{"a":1, "b":2}');
 jsonb_ne
----------
 t
(1 row)
```


- jsonb_gt(jsonb, jsonb)

  Description: Same as the operator **`>`**, compares two values.

  Return type: Boolean

  Example:

```
gaussdb=# SELECT jsonb_gt('["a", "b"]', '{"a":1, "b":2}');
 jsonb_gt
----------
 f
(1 row)
```

- jsonb_ge(jsonb, jsonb)

  Description: Same as the operator **`>=`**, compares two values.

  Return type: Boolean

  Example:

```
gaussdb=# SELECT jsonb_ge('["a", "b"]', '{"a":1, "b":2}');
 jsonb_ge
----------
 f
(1 row)
```

- jsonb_lt(jsonb, jsonb)

  Description: Same as the operator **`<`**, compares two values.

  Return type: Boolean

  Example:

```
gaussdb=# SELECT jsonb_lt('["a", "b"]', '{"a":1, "b":2}');
 jsonb_lt
----------
 t
(1 row)
```

- jsonb_le(jsonb, jsonb)

  Description: Same as the operator **`<=`**, compares two values.

  Return type: Boolean

  Example:

```
gaussdb=# SELECT jsonb_le('["a", "b"]', '{"a":1, "b":2}');
 jsonb_le
----------
 t
(1 row)
```

- to_json(anyelement)

  Description: Converts parameters to **`json`**.

  Return type: json

  Example:

```
gaussdb=# SELECT to_json('{1,5}'::text[]);
   to_json
-----------
 ["1","5"]
(1 row)
```

- to_jsonb(anyelement)

  Description: Converts the input anyelement parameter to the JSONB type.

  Return type: jsonb

NOTICE

- When the parameter is an empty string (''), if the SQL compatibility mode of the database is A, **NULL** is returned; if the SQL compatibility mode of the database is PG, an empty string is returned. This is because empty strings are treated as **NULL** in A compatibility mode.
- When the parameter is of the DATE type, if the database SQL compatibility mode is A, the returned result contains hour, minute, and second; if the database SQL compatibility mode is PG, the returned result does not contain hour, minute, and second. This is because DATE will be replaced by TIMESTAMP(0) WITHOUT TIME ZONE in A compatibility mode.

Example:

```
gaussdb=# SELECT to_jsonb(ARRAY[1, 2, 3, 4]);
  to_jsonb
--------------
 [1, 2, 3, 4]
(1 row)
```

- jsonb_hash(jsonb)

  Description: Performs the hash operation on JSONB.

  Return type: integer

  Example:

```
gaussdb=# SELECT jsonb_hash('[1,2,3]');
 jsonb_hash
------------
 -559968547
(1 row)
```

- Other Functions

  Description: Internal functions used by JSON and JSONB aggregate functions.

```
json_agg_transfn
json_agg_finalfn
json_object_agg_transfn
json_object_agg_finalfn
```

7.6.14 HLL Functions and Operators

Hash Functions

- hll_hash_boolean(bool)

  Description: Hashes data of the Boolean type.

  Return type: hll_hashval

  Example:

```
gaussdb=# SELECT hll_hash_boolean(FALSE);
 hll_hash_boolean
---------------------
 -5451962507482445012
(1 row)
```

- hll_hash_boolean(bool, int32)

  Description: Configures a hash seed (that is, change the hash policy) and hashes data of the Boolean type.

  Return type: hll_hashval

Example:

```
gaussdb=# SELECT hll_hash_boolean(FALSE, 10);
  hll_hash_boolean
--------------------
 -1169037589280886076
(1 row)
```

- hll_hash_smallint(smallint)

  Description: Hashes data of the smallint type.

  Return type: hll_hashval

  Example:

```
gaussdb=# SELECT hll_hash_smallint(100::smallint);
  hll_hash_smallint
---------------------
 962727970174027904
(1 row)
```

NOTE

If parameters with the same numeric value are hashed using different data types, the data will differ, because hash functions select different calculation policies for each type.

- hll_hash_smallint(smallint, int32)

  Description: Configures a hash seed (that is, change the hash policy) and hashes data of the smallint type.

  Return type: hll_hashval

  Example:

```
gaussdb=# SELECT hll_hash_smallint(100::smallint, 10);
  hll_hash_smallint
---------------------
 -9056177146160443041
(1 row)
```

- hll_hash_integer(integer)

  Description: Hashes data of the integer type.

  Return type: hll_hashval

  Example:

```
gaussdb=# SELECT hll_hash_integer(0);
   hll_hash_integer
----------------------
 5156626420896634997
(1 row)
```

- hll_hash_integer(integer, int32)

  Description: Hashes data of the integer type and configures a hash seed (that is, change the hash policy).

  Return type: hll_hashval

  Example:

```
gaussdb=# SELECT hll_hash_integer(0, 10);
  hll_hash_integer
--------------------
 -5035020264353794276
(1 row)
```

- hll_hash_bigint(bigint)

  Description: Hashes data of the bigint type.

  Return type: hll_hashval

Example:

```
gaussdb=# SELECT hll_hash_bigint(100::bigint);
   hll_hash_bigint
---------------------
 -2401963681423227794
(1 row)
```

- hll_hash_bigint(bigint, int32)

  Description: Hashes data of the bigint type and configures a hash seed (that is, changes the hash policy).

  Return type: hll_hashval

  Example:

```
gaussdb=# SELECT hll_hash_bigint(100::bigint, 10);
   hll_hash_bigint
---------------------
 -2305749404374433531
(1 row)
```

- hll_hash_bytea(bytea)

  Description: Hashes data of the bytea type.

  Return type: hll_hashval

  Example:

```
gaussdb=# SELECT hll_hash_bytea(E'\\x');
 hll_hash_bytea
----------------
 0
(1 row)
```

- hll_hash_bytea(bytea, int32)

  Description: Hashes data of the bytea type and configures a hash seed (that is, changes the hash policy).

  Return type: hll_hashval

  Example:

```
gaussdb=# SELECT hll_hash_bytea(E'\\x', 10);
   hll_hash_bytea
---------------------
 7233188113542599437
(1 row)
```

- hll_hash_text(text)

  Description: Hashes data of the text type.

  Return type: hll_hashval

  Example:

```
gaussdb=# SELECT hll_hash_text('AB');
    hll_hash_text
---------------------
 -5666002586880275174
(1 row)
```

- hll_hash_text(text, int32)

  Description: Hashes data of the text type and configures a hash seed (that is, changes the hash policy).

  Return type: hll_hashval

  Example:

```
gaussdb=# SELECT hll_hash_text('AB', 10);
hll_hash_text
---------------------
```



```
-2215507121143724132
(1 row)
```

- hll_hash_any(anytype)

  Description: Hashes data of any type.

  Return type: hll_hashval

  Example:

```
gaussdb=# SELECT hll_hash_any(1);
    hll_hash_any
----------------------
 -1316670585935156930
(1 row)

gaussdb=# SELECT hll_hash_any('08:00:2b:01:02:03'::macaddr);
    hll_hash_any
----------------------
 -3719950434455589360
(1 row)
```

- hll_hash_any(anytype, int32)

  Description: Hashes data of any type and configures a hash seed (that is, changes the hash policy).

  Return type: hll_hashval

  Example:

```
gaussdb=# SELECT hll_hash_any(1, 10);
    hll_hash_any
----------------------
 7048553517657992351
(1 row)
```

- hll_hashval_eq(hll_hashval, hll_hashval)

  Description: Compares two pieces of data of the hll_hashval type to check whether they are the same.

  Return type: Boolean

  Example:

```
gaussdb=# SELECT hll_hashval_eq(hll_hash_integer(1), hll_hash_integer(1));
 hll_hashval_eq
----------------
 t
(1 row)
```

- hll_hashval_ne(hll_hashval, hll_hashval)

  Description: Compares two pieces of data of the hll_hashval type to check whether they are different.

  Return type: Boolean

  Example:

```
gaussdb=# SELECT hll_hashval_ne(hll_hash_integer(1), hll_hash_integer(1));
 hll_hashval_ne
----------------
 f
(1 row)
```

HLL Functions

There are three HLL modes: explicit, sparse, and full. When the data size is small, the explicit mode is used. In this mode, distinct values are calculated without errors. As the number of distinct values increases, the HLL mode is switched to the sparse and full modes in sequence. The two modes have no difference in the

calculation result, but vary in the calculation efficiency of HLL functions and the storage space of HLL objects. The following functions can be used to view some HLL parameters:

- hll_print(hll)

  Description: Prints some debugging parameters of an HLL.

  Example:

```
gaussdb=# SELECT hll_print(hll_empty());
                                hll_print
--------------------------------------------------------------------------------
 type=1(HLL_EMPTY), log2m=14, log2explicit=10, log2sparse=12, duplicatecheck=0
(1 row)
```

- hll_type(hll)

  Description: Checks the type of the current HLL. The return values are described as follows: **0** indicates **HLL_UNINIT**, an HLL object that is not initialized. **1** indicates **HLL_EMPTY**, an empty HLL object. **2** indicates **HLL_EXPLICIT**, an HLL object in explicit mode. **3** indicates **HLL_SPARSE**, an HLL object in sparse mode. **4** indicates **HLL_FULL**, an HLL object in full mode. **5** indicates **HLL_UNDEFINED**, an invalid HLL object.

  Example:

```
gaussdb=# SELECT hll_type(hll_empty());
 hll_type
----------
        1
(1 row)
```

- hll_log2m(hll)

  Description: Checks the value of **log2m** in the current HLL data structure. **log2m** is the logarithm of the number of buckets. This value affects the error rate of calculating distinct values by HLL. The error rate = ±1.04/√(2^log2m). When the value of **log2m** is explicitly set to a value ranging from 10 to 16, HLL sets the number of buckets to 2^{log2m}. When the value of **log2explicit** is explicitly set to **–1**, the built-in default value is used.

  Example:

```
gaussdb=# SELECT hll_log2m(hll_empty());
 hll_log2m
-----------
        14
(1 row)

gaussdb=# SELECT hll_log2m(hll_empty(10));
 hll_log2m
-----------
        10
(1 row)

gaussdb=# SELECT hll_log2m(hll_empty(-1));
 hll_log2m
-----------
        14
(1 row)
```

- hll_log2explicit(hll)

  Description: Queries the value of **log2explicit** in the current HLL data structure. Generally, the HLL changes from the explicit mode to the sparse mode and then to the full mode. This process is called the promotion hierarchy policy. You can change the value of **log2explicit** to change the policy. For example, if the value of **log2explicit** is **0**, the HLL will skip the

explicit mode and directly enter the sparse mode. When the value of **log2explicit** is explicitly set to a value ranging from 1 to 12, the HLL will switch to the sparse mode when the length of the data segment exceeds $2^{log2explicit}$. When the value of **log2explicit** is explicitly set to **–1**, the built-in default value is used.

Example:

```
gaussdb=# SELECT hll_log2explicit(hll_empty());
 hll_log2explicit
------------------
               10
(1 row)

gaussdb=# SELECT hll_log2explicit(hll_empty(12, 8));
 hll_log2explicit
------------------
                8
(1 row)

gaussdb=# SELECT hll_log2explicit(hll_empty(12, -1));
 hll_log2explicit
------------------
               10
(1 row)
```

- hll_log2sparse(hll)

  Description: Queries the value of **log2sparse** in the current HLL data structure. Generally, the HLL changes from the explicit mode to the sparse mode and then to the full mode. This process is called the promotion hierarchy policy. You can adjust the value of **log2sparse** to change the policy. For example, if the value of **log2sparse** is **0**, the system skips the sparse mode and directly enters the full mode. If the value of **log2sparse** is explicitly set to a value ranging from 1 to 14, the HLL will switch to the full mode when the length of the data segment exceeds $2^{log2sparse}$. When the value of **log2sparse** is explicitly set to **–1**, the built-in default value is used.

  Example:

  ```
  gaussdb=# SELECT hll_log2sparse(hll_empty());
   hll_log2sparse
  ----------------
              12
  (1 row)

  gaussdb=# SELECT hll_log2sparse(hll_empty(12, 8, 10));
   hll_log2sparse
  ----------------
              10
  (1 row)

  gaussdb=# SELECT hll_log2sparse(hll_empty(12, 8, -1));
   hll_log2sparse
  ----------------
              12
  (1 row)
  ```

- hll_duplicatecheck(hll)

  Description: Specifies whether duplicate check is enabled. **0**: disable; **1**: enable. This function is disabled by default. If there are many duplicate values, you can enable this function to improve efficiency. When the value of **duplicatecheck** is explicitly set to **–1**, the built-in default value is used.

  Example:

  ```
  gaussdb=# SELECT hll_duplicatecheck(hll_empty());
   hll_duplicatecheck
  ```



```
--------------------
                0
(1 row)

gaussdb=# SELECT hll_duplicatecheck(hll_empty(12, 8, 10, 1));
 hll_duplicatecheck
--------------------
                1
(1 row)

gaussdb=# SELECT hll_duplicatecheck(hll_empty(12, 8, 10, -1));
 hll_duplicatecheck
--------------------
                0
(1 row)
```

Functional Functions

- hll_empty()

  Description: Creates an empty HLL.

  Return type: HLL

  Example:

  ```
  gaussdb=# SELECT hll_empty();
                          hll_empty
  ------------------------------------------------------------
   \x484c4c00000000002b050000000000000000000000000000000000000000
  (1 row)
  ```

- hll_empty(int32 log2m)

  Description: Creates an empty HLL and sets the **log2m** parameter. The parameter value ranges from 10 to 16. If the input is **–1**, the built-in default value is used.

  Return type: HLL

  Example:

  ```
  gaussdb=# SELECT hll_empty(10);
                          hll_empty
  ------------------------------------------------------------
   \x484c4c00000000002b040000000000000000000000000000000000000000
  (1 row)

  gaussdb=# SELECT hll_empty(-1);
                          hll_empty
  ------------------------------------------------------------
   \x484c4c00000000002b050000000000000000000000000000000000000000
  (1 row)
  ```

- hll_empty(int32 log2m, int32 log2explicit)

  Description: Creates an empty HLL and sets the **log2m** and **log2explicit** parameters in sequence. The value of **log2explicit** ranges from 0 to 12. The value **0** indicates that the explicit mode is skipped. This parameter is used to set the threshold of the explicit mode. When the length of the data segment reaches $2^{log2explicit}$, the mode is switched to the sparse or full mode. If the input is **–1**, the built-in default value of **log2explicit** is used.

  Return type: HLL

  Example:

  ```
  gaussdb=# SELECT hll_empty(10, 4);
                          hll_empty
  ------------------------------------------------------------
   \x484c4c0000000000130400000000000000000000000000000000000000
  ```



```
(1 row)

gaussdb=# SELECT hll_empty(10, -1);
                                                            hll_empty
------------------------------------------------------------------------------------------------------------------
 \x484c4c00000000002b040000000000000000000000000000000000000000
(1 row)
```

- hll_empty(int32 log2m, int32 log2explicit, int64 log2sparse)

  Description: Creates an empty HLL and sets the **log2m**, **log2explicit** and **log2sparse** parameters in sequence. The value of **log2sparse** ranges from 0 to 14. The value **0** indicates that the sparse mode is skipped. This parameter is used to set the threshold of the sparse mode. When the length of the data segment reaches $2^{log2sparse}$, the mode is switched to the full mode. If the input is **–1**, the built-in default value of **log2sparse** is used.

  Return type: HLL

  Example:

```
gaussdb=# SELECT hll_empty(10, 4, 8);
                                                            hll_empty
------------------------------------------------------------------------------------------------------------------
 \x484c4c0000000000120400000000000000000000000000000000000000000000
(1 row)

gaussdb=# SELECT hll_empty(10, 4, -1);
                                                            hll_empty
------------------------------------------------------------------------------------------------------------------
 \x484c4c0000000000130400000000000000000000000000000000000000000000
(1 row)
```

- hll_empty(int32 log2m, int32 log2explicit, int64 log2sparse, int32 duplicatecheck)

  Description: Creates an empty HLL and sets the **log2m**, **log2explicit**, **log2sparse**, and **duplicatecheck** parameters in sequence. The value of **duplicatecheck** is **0** or **1**, indicating whether the duplicate check mode is enabled. By default, this mode is disabled. If the input is **–1**, the built-in default value of **duplicatecheck** is used.

  Return type: HLL

  Example:

```
gaussdb=# SELECT hll_empty(10, 4, 8, 0);
                                                            hll_empty
------------------------------------------------------------------------------------------------------------------
 \x484c4c0000000000120400000000000000000000000000000000000000000000
(1 row)

gaussdb=# SELECT hll_empty(10, 4, 8, -1);
                                                            hll_empty
------------------------------------------------------------------------------------------------------------------
 \x484c4c0000000000120400000000000000000000000000000000000000000000
(1 row)
```

- hll_add(hll, hll_hashval)

  Description: Adds hll_hashval to an HLL.

  Return type: HLL

  Example:

```
gaussdb=# SELECT hll_add(hll_empty(), hll_hash_integer(1));
                                                                  hll_add
------------------------------------------------------------------------------------------------------------------------------
 \x484c4c08000002002b0900000000000000f03f3e2921ff133fbaed3e2921ff133fbaed00
(1 row)
```


- hll_add_rev(hll_hashval, hll)

  Description: Adds **hll_hashval** to an HLL. This function works the same as **hll_add**, except that the positions of parameters are switched.

  Return type: HLL

  Example:

```
gaussdb=# SELECT hll_add_rev(hll_hash_integer(1), hll_empty());
                              hll_add_rev
------------------------------------------------------------------------------
 \x484c4c08000002002b0900000000000000f03f3e2921ff133fbaed3e2921ff133fbaed00
(1 row)
```

- hll_eq(hll, hll)

  Description: Compares two HLLs to check whether they are the same.

  Return type: Boolean

  Example:

```
gaussdb=# SELECT hll_eq(hll_add(hll_empty(), hll_hash_integer(1)), hll_add(hll_empty(),
hll_hash_integer(2)));
 hll_eq
--------
 f
(1 row)
```

- hll_ne(hll, hll)

  Description: Compares two HLLs to check whether they are different.

  Return type: Boolean

  Example:

```
gaussdb=# SELECT hll_ne(hll_add(hll_empty(), hll_hash_integer(1)), hll_add(hll_empty(),
hll_hash_integer(2)));
 hll_ne
--------
 t
(1 row)
```

- hll_cardinality(hll)

  Description: Calculates the number of distinct values of an HLL.

  Return type: int

  Example:

```
gaussdb=# SELECT hll_cardinality(hll_empty() || hll_hash_integer(1));
 hll_cardinality
-----------------
               1
(1 row)
```

- hll_union(hll, hll)

  Description: Performs the UNION operation on two HLL data structures to obtain one HLL.

  Return type: HLL

  Example:

```
gaussdb=# SELECT hll_union(hll_add(hll_empty(), hll_hash_integer(1)), hll_add(hll_empty(),
hll_hash_integer(2)));
                                    hll_union
------------------------------------------------------------------------------------------

\x484c4c10002000002b0900000000000000004000000000000000000b3ccc49320cca1ae3e2921ff133fba
ed00
(1 row)
```


Aggregate Functions

- hll_add_agg(hll_hashval)

  Description: Groups hashed data into HLL.

  Return type: HLL

  Example:

```
-- Prepare data:
gaussdb=# CREATE TABLE t_id(id int);
gaussdb=# INSERT INTO t_id VALUES(generate_series(1,500));
gaussdb=# CREATE TABLE t_data(a int, c text);
gaussdb=# INSERT INTO t_data SELECT mod(id,2), id FROM t_id;

-- Create a table and specify an HLL column:
gaussdb=# CREATE TABLE t_a_c_hll(a int, c hll);

-- Use GROUP BY on column a to group data, and insert the data to the HLL:
gaussdb=# INSERT INTO t_a_c_hll SELECT a, hll_add_agg(hll_hash_text(c)) FROM t_data GROUP BY a;

-- Calculate the number of distinct values for each group in the HLL:
gaussdb=# SELECT a, #c AS cardinality FROM t_a_c_hll ORDER BY a;
 a |   cardinality
---+------------------
 0 | 247.862354346299
 1 | 250.908710610377
(2 rows)
```

- hll_add_agg(hll_hashval, int32 log2m)

  Description: Groups hashed data into HLL and specifies the **log2m** parameter. The value ranges from 10 to 16. If the input is **–1** or **NULL**, the built-in default value is used.

  Return type: HLL

  Example:

```
gaussdb=# SELECT hll_cardinality(hll_add_agg(hll_hash_text(c), 12)) FROM t_data;
 hll_cardinality
------------------
 497.965240179228
(1 row)
```

- hll_add_agg(hll_hashval, int32 log2m, int32 log2explicit)

  Description: Groups hashed data into HLL and specifies the **log2m** and **log2explicit** parameters in sequence. The value of **log2explicit** ranges from 0 to 12. The value **0** indicates that the explicit mode is skipped. This parameter is used to set the threshold of the explicit mode. When the length of the data segment reaches $2^{log2explicit}$, the mode is switched to the sparse or full mode. If the input is **–1** or **NULL**, the built-in default value of **log2explicit** is used.

  Return type: HLL

  Example:

```
gaussdb=# SELECT hll_cardinality(hll_add_agg(hll_hash_text(c), NULL, 1)) FROM t_data;
 hll_cardinality
------------------
 498.496062953313
(1 row)
```

- hll_add_agg(hll_hashval, int32 log2m, int32 log2explicit, int64 log2sparse)

  Description: Groups hashed data into HLL and sets the **log2m**, **log2explicit**, and **log2sparse** parameters in sequence. The value of **log2sparse** ranges from 0 to 14. The value **0** indicates that the sparse mode is skipped. This parameter is used to set the threshold of the sparse mode. When the length of the data

segment reaches $2^{log2sparse}$, the mode is switched to the full mode. If the input is **–1** or **NULL**, the built-in default value of **log2sparse** is used.

Return type: HLL

Example:

```
gaussdb=# SELECT hll_cardinality(hll_add_agg(hll_hash_text(c), NULL, 6, 10)) FROM t_data;
 hll_cardinality
------------------
 498.496062953313
(1 row)
```

- hll_add_agg(hll_hashval, int32 log2m, int32 log2explicit, int64 log2sparse, int32 duplicatecheck)

  Description: Groups hashed data into HLL and sets the **log2m**, **log2explicit**, **log2sparse**, and **duplicatecheck** parameters. The value of **duplicatecheck** can be **0** or **1**, indicating whether to enable this mode. By default, this mode is disabled. If the input is **–1** or **NULL**, the built-in default value of **duplicatecheck** is used.

  Return type: HLL

  Example:

  ```
  gaussdb=# SELECT hll_cardinality(hll_add_agg(hll_hash_text(c), NULL, 6, 10, -1)) FROM t_data;
   hll_cardinality
  ------------------
   498.496062953313
  (1 row)
  ```

- hll_union_agg(hll)

  Description: Performs the UNION operation on multiple pieces of data of the HLL type to obtain one HLL.

  Return type: HLL

  Example:

  ```
  -- Perform the UNION operation on data of the HLL type in each group to obtain one HLL, and calculate the number of distinct values:
  gaussdb=# SELECT #hll_union_agg(c) AS cardinality FROM t_a_c_hll;
    cardinality
  ------------------
   498.496062953313
  (1 row)
  ```

  NOTE

  To perform the UNION operation on data in multiple HLLs, ensure that the HLLs have the same precision. Otherwise, UNION cannot be performed. This constraint also applies to the hll_union(hll, hll) function.

Obsolete Functions

Some old HLL functions are discarded due to version upgrade. You can replace them with similar functions.

- hll_schema_version(hll)

  Description: Checks the schema version in the current HLL. In earlier versions, the schema version is fixed at **1**, which is used to verify the header of the HLL field. After refactoring, the HLL field is added to the header for verification. The schema version is no longer used.

- hll_regwidth(hll)

Description: Queries the bucket size in the HLL data structure. In earlier versions, the value of **regwidth** ranges from 1 to 5, which has a large error and limits the upper limit of the cardinality estimation. After refactoring, the value of **regwidth** is fixed at **6** and the variable is not used.

- hll_expthresh(hll)

  Description: Obtains the value of **expthresh** in the current HLL. The hll_log2explicit(hll) function is used to replace similar functions.

- hll_sparseon(hll)

  Description: Specifies whether the sparse mode is enabled. Use hll_log2sparse(hll) to replace similar functions. The value **0** indicates that the sparse mode is disabled.

Built-In Functions

HLL has a series of built-in functions for internal data processing. Generally, users do not need to know how to use these functions. For details, see **Table 7-77**.

Table 7-77 Built-In Functions

Function	Description
hll_in	Receives HLL data in string format.
hll_out	Sends HLL data in string format.
hll_recv	Receives HLL data in bytea format.
hll_send	Sends HLL data in bytea format.
hll_trans_in	Receives hll_trans_type data in string format.
hll_trans_out	Sends hll_trans_type data in string format.
hll_trans_recv	Receives hll_trans_type data in bytea format.
hll_trans_send	Sends hll_trans_type data in bytea format.
hll_typmod_in	Receives typmod data.
hll_typmod_out	Sends typmod data.
hll_hashval_in	Receives hll_hashval data.
hll_hashval_out	Sends hll_hashval data.
hll_add_trans0	Works similar to hll_add. No input parameter is specified during initialization. It is usually used on DNs in the first phase of aggregation operations.
hll_add_trans1	Works similar to hll_add. An input parameter is specified during initialization. It is usually used on DNs in the first phase of aggregation operations.
hll_add_trans2	Works similar to hll_add. Two input parameters are specified during initialization. It is usually used on DNs in the first phase of aggregation operations.

Function	Description
hll_add_trans3	Works similar to hll_add. Three input parameters are specified during initialization. It is usually used on DNs in the first phase of aggregation operations.
hll_add_trans4	Works similar to hll_add. Four input parameters are specified during initialization. It is usually used on DNs in the first phase of aggregation operations.
hll_union_trans	Works similar to hll_union and is used on DNs in the first phase of aggregation operations.
hll_union_collect	Works similar to hll_union and is used on DNs in the second phase of aggregation operations to summarize the results of each DN.
hll_pack	Converts the user-defined hll_trans_type to the HLL type on DNs in the third phase of aggregation operations.
hll	Converts an HLL type to another HLL type. Input parameters can be specified.
hll_hashval	Converts the bigint type to the hll_hashval type.
hll_hashval_int4	Converts the int4 type to the hll_hashval type.

Operators

- =

  Description: Compares the values of the HLL or hll_hashval type to check whether they are the same.

  Return type: Boolean

  Example:

```
--hll
gaussdb=# SELECT (hll_empty() || hll_hash_integer(1)) = (hll_empty() || hll_hash_integer(1));
column
----------
 t
(1 row)

--hll_hashval
gaussdb=# SELECT hll_hash_integer(1) = hll_hash_integer(1);
 ?column?
----------
 t
(1 row)
```

- <> or !=

  Description: Compares the values of the HLL or hll_hashval type to check whether they are different.

  Return type: Boolean

  Example:

```
--hll
gaussdb=# SELECT (hll_empty() || hll_hash_integer(1)) <> (hll_empty() || hll_hash_integer(2));
 ?column?
```



```
----------
 t
(1 row)

--hll_hashval
gaussdb=# SELECT hll_hash_integer(1) <> hll_hash_integer(2);
 ?column?
----------
 t
(1 row)
```

- ||

  Description: Represents the hll_add, hll_union, and hll_add_rev functions.

  Return type: HLL

  Example:

```
--hll_add
gaussdb=# SELECT hll_empty() || hll_hash_integer(1);
                                ?column?
--------------------------------------------------------------------------------
 \x484c4c08000002002b0900000000000000f03f3e2921ff133fbaed3e2921ff133fbaed00
(1 row)

--hll_add_rev
gaussdb=# SELECT hll_hash_integer(1) || hll_empty();
                                ?column?
--------------------------------------------------------------------------------
 \x484c4c08000002002b0900000000000000f03f3e2921ff133fbaed3e2921ff133fbaed00
(1 row)

--hll_union
gaussdb=# SELECT (hll_empty() || hll_hash_integer(1)) || (hll_empty() || hll_hash_integer(2));
                                      ?column?
-----------------------------------------------------------------------------------------------

\x484c4c10002000002b0900000000000000004000000000000000000b3ccc49320cca1ae3e2921ff133fba
ed00
(1 row)
```

- #

  Description: Calculates the number of distinct values of an HLL. It works the same as the hll_cardinality function.

  Return type: int

  Example:

```
gaussdb=# SELECT #(hll_empty() || hll_hash_integer(1));
 ?column?
----------
        1
(1 row)
```

7.6.15 SEQUENCE Functions

The sequence functions provide a simple method to ensure security of multiple users for users to obtain sequence values from sequence objects.

- nextval(regclass)

  Description: Specifies an increasing sequence and returns a new value.

> **NOTE**
>
> To avoid blocking of concurrent transactions that obtain numbers from the same sequence, a nextval operation is never rolled back; that is, once a value is fetched, it is considered used, even if the transaction that did the nextval later aborts. This means that aborted transactions may leave unused "holes" in the sequence of assigned values. Therefore, sequences in GaussDB cannot be used to obtain sequence without gaps.

NOTICE

The **nextval** function can be executed only on the primary node. It is not supported on standby nodes.

Return type: numeric

The nextval function can be called in either of the following ways: (In example 2, the sequence name cannot contain a dot.)

```
gaussdb=# CREATE SEQUENCE seqDemo;
-- Example 1:
gaussdb=# SELECT nextval('seqDemo');
 nextval
---------
       1
(1 row)
-- Example 2:
gaussdb=# SELECT seqDemo.nextval;
 nextval
---------
       2
(1 row)
gaussdb=# DROP SEQUENCE seqDemo;
```

- currval(regclass)

  Description: Returns the last value of **nextval** in the current session. If nextval has not been called for the specified sequence in the current session, an error is reported when currval is called.

  Return type: numeric

  The currval function can be called in either of the following ways: (In example 2, the sequence name cannot contain a dot.)

```
gaussdb=# CREATE SEQUENCE seq1;
gaussdb=# SELECT nextval('seq1');
-- Example 1:
gaussdb=# SELECT currval('seq1');
 currval
---------
       1
(1 row)
-- Example 2:
gaussdb=# SELECT seq1.currval;
 currval
---------
       1
(1 row)
gaussdb=# DROP SEQUENCE seq1;
```

- lastval()

  Description: Returns the last value of **nextval** in the current session. This function is equivalent to **currval**, except that it does not use sequence names as parameters. It fetches the sequence used by **nextval** last time in the

current session. If nextval has not been called in the current session, an error is reported when lastval is called.

Return type: numeric

Example:

```
gaussdb=# CREATE SEQUENCE seq1;
gaussdb=# SELECT nextval('seq1');
gaussdb=# SELECT lastval();
 lastval
---------
       1
(1 row)
gaussdb=# DROP SEQUENCE seq1;
```

- setval(regclass, numeric)

  Sets the current value of a sequence.

  Return type: numeric

  Example:

  ```
  gaussdb=# CREATE SEQUENCE seqDemo;
  gaussdb=# SELECT nextval('seqDemo');
  gaussdb=# SELECT setval('seqDemo',5);
   setval
  --------
       5
  (1 row)
  gaussdb=# DROP SEQUENCE seqDemo;
  ```

- setval(regclass, numeric, Boolean)

  Sets the current value of a sequence and the is_called sign.

  Return type: numeric

  Example:

  ```
  gaussdb=# CREATE SEQUENCE seqDemo;
  gaussdb=# SELECT nextval('seqDemo');
  gaussdb=# SELECT setval('seqDemo',5,true);
   setval
  --------
       5
  (1 row)
  gaussdb=# DROP SEQUENCE seqDemo;
  ```

  NOTE

  The current session will take effect immediately after **setval** is performed. If other sessions have buffered sequence values, **setval** will take effect only after the values are used up. Therefore, to prevent sequence value conflicts, you are advised to use **setval** with caution.

  Because the sequence is non-transactional, changes made by **setval** will not be canceled when a transaction rolled back.

NOTICE

The **nextval** function can be executed only on the primary node. It is not supported on standby nodes.

- pg_sequence_last_value(sequence_oid oid, OUT cache_value int16, OUT last_value int16)

  Description: Obtains the parameters of a specified sequence, including the cache value and current value.

Return type: int16, int16

- last_insert_id()

  Description: Gets the first auto-generated value that was last successfully inserted for an auto-increment column.

  Return type: int16

  Example:

```
gaussdb=# CREATE TABLE animals_test (
gaussdb(# id int PRIMARY KEY NOT NULL AUTO_INCREMENT,
gaussdb(# name CHAR(30) NOT NULL
gaussdb(# );
NOTICE:  CREATE TABLE will create implicit sequence "animals_test_id_seq" for serial column "animals_test.id"
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "animals_test_pkey" for table "animals_test"
CREATE TABLE
gaussdb=# SELECT last_insert_id();
 last_insert_id
----------------
              0
(1 row)
gaussdb=# INSERT INTO animals_test (name) VALUES ('dog');
INSERT 0 1
gaussdb=# SELECT last_insert_id();
 last_insert_id
----------------
              1
(1 row)
```

- last_insert_id(int16)

  Description: Sets the return value of the next last_insert_id() function and returns the value. If the parameter is **NULL**, set the return value of the next last_insert_id() function to **0**. This function returns **NULL**.

  Return type: int16

  Example:

```
gaussdb=# SELECT last_insert_id(100);
 last_insert_id
----------------
            100
(1 row)
gaussdb=# SELECT last_insert_id();
 last_insert_id
----------------
            100
(1 row)
```

  NOTE

  - last_insert_id() and last_insert_id(int16) are session-level functions. If no data is inserted into the auto-increment column in the current session, last_insert_id() returns **0**.
  - last_insert_id() and last_insert_id(int16) are available only when **sql_compatibility** is set to **'B'**.

7.6.16 Array Functions and Operators

Array Operators

- =

  Description: Specifies whether two arrays are equal.

Example:

```
gaussdb=# SELECT ARRAY[1.1,2.1,3.1]::int[] = ARRAY[1,2,3] AS RESULT ;
 result
--------
 t
(1 row)
```

- <>

  Description: Specifies whether two arrays are not equal.

  Example:

  ```
  gaussdb=# SELECT ARRAY[1,2,3] <> ARRAY[1,2,4] AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- <

  Description: Specifies whether an array is less than another.

  Example:

  ```
  gaussdb=# SELECT ARRAY[1,2,3] < ARRAY[1,2,4] AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- >

  Description: Specifies whether an array is greater than another.

  Example:

  ```
  gaussdb=# SELECT ARRAY[1,4,3] > ARRAY[1,2,4] AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- <=

  Description: Specifies whether an array is less than another.

  Example:

  ```
  gaussdb=# SELECT ARRAY[1,2,3] <= ARRAY[1,2,3] AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- >=

  Description: Specifies whether an array is greater than or equal to another.

  Example:

  ```
  gaussdb=# SELECT ARRAY[1,4,3] >= ARRAY[1,4,3] AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- @>

  Description: Specifies whether an array contains another.

  Example:

  ```
  gaussdb=# SELECT ARRAY[1,4,3] @> ARRAY[3,1] AS RESULT;
   result
  --------
   t
  (1 row)
  ```


- <@

  Description: Specifies whether an array is contained in another.

  Example:

```
gaussdb=# SELECT ARRAY[2,7] <@ ARRAY[1,7,4,2,6] AS RESULT;
 result
--------
 t
(1 row)
```

- &&

  Description: Specifies whether an array overlaps another (have common elements).

  Example:

```
gaussdb=# SELECT ARRAY[1,4,3] && ARRAY[2,1] AS RESULT;
 result
--------
 t
(1 row)
```

- ||

  Description: Array-to-array concatenation

  Example:

```
gaussdb=# SELECT ARRAY[1,2,3] || ARRAY[4,5,6] AS RESULT;
    result
---------------
 {1,2,3,4,5,6}
(1 row)
gaussdb=# SELECT ARRAY[1,2,3] || ARRAY[[4,5,6],[7,8,9]] AS RESULT;
          result
---------------------------
 {{1,2,3},{4,5,6},{7,8,9}}
(1 row)
```

- ||

  Description: Element-to-array concatenation

  Example:

```
gaussdb=# SELECT 3 || ARRAY[4,5,6] AS RESULT;
  result
-----------
 {3,4,5,6}
(1 row)
```

- ||

  Description: Array-to-element concatenation

  Example:

```
gaussdb=# SELECT ARRAY[4,5,6] || 7 AS RESULT;
  result
-----------
 {4,5,6,7}
(1 row)
```

Array comparisons compare the array contents element-by-element, using the default B-tree comparison function for the element data type. In multidimensional arrays, the elements are accessed in row-major order. If the contents of two arrays are equal but the dimensionality is different, the first difference in the dimensionality information determines the sort order.

Array Functions

- array_append(anyarray, anyelement)

  Description: Appends an element to the end of an array, and only supports one-dimensional arrays.

  Return type: anyarray

  Example:

```
gaussdb=# SELECT array_append(ARRAY[1,2], 3) AS RESULT;
 result
---------
 {1,2,3}
(1 row)
```

- array_prepend(anyelement, anyarray)

  Description: Appends an element to the beginning of an array, and only supports one-dimensional arrays.

  Return type: anyarray

  Example:

```
gaussdb=# SELECT array_prepend(1, ARRAY[2,3]) AS RESULT;
 result
---------
 {1,2,3}
(1 row)
```

- array_cat(anyarray, anyarray)

  Description: Concatenates two arrays, and supports multi-dimensional arrays.

  Return type: anyarray

  Example:

```
gaussdb=# SELECT array_cat(ARRAY[1,2,3], ARRAY[4,5]) AS RESULT;
   result
-------------
 {1,2,3,4,5}
(1 row)

gaussdb=# SELECT array_cat(ARRAY[[1,2],[4,5]], ARRAY[6,7]) AS RESULT;
      result
---------------------
 {{1,2},{4,5},{6,7}}
(1 row)
```

- array_union(anyarray, anyarray)

  Description: Concatenates two arrays. Only one-dimensional arrays are supported. If an input parameter is **NULL**, another input parameter is returned.

  Return type: anyarray

  Example:

```
gaussdb=# SELECT array_union(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
   result
-------------
 {1,2,3,3,4,5}
(1 row)

gaussdb=# SELECT array_union(ARRAY[1,2,3], NULL) AS RESULT;
 result
---------
 {1,2,3}
(1 row)
```

- array_union_distinct(anyarray, anyarray)

Description: Concatenates two arrays and deduplicates them. Only one-dimensional arrays are supported. If an input parameter is **NULL**, another input parameter is returned.

Return type: anyarray

Example:

```
gaussdb=# SELECT array_union_distinct(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
   result
-------------
 {1,2,3,4,5}
(1 row)

gaussdb=# SELECT array_union_distinct(ARRAY[1,2,3], NULL) AS RESULT;
 result
---------
 {1,2,3}
(1 row)
```

- array_intersect(anyarray, anyarray)

  Description: Intersects two arrays. Only one-dimensional arrays are supported. If any input parameter is **NULL**, **NULL** is returned.

  Return type: anyarray

  Example:

  ```
  gaussdb=# SELECT array_intersect(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
     result
  -------------
   {3}
  (1 row)

  gaussdb=# SELECT array_intersect(ARRAY[1,2,3], NULL) AS RESULT;
   result
  --------

  (1 row)
  ```

- array_intersect_distinct(anyarray, anyarray)

  Description: Intersects two arrays and deduplicates them. Only one-dimensional arrays are supported. If any input parameter is **NULL**, **NULL** is returned.

  Return type: anyarray

  Example:

  ```
  gaussdb=# SELECT array_intersect_distinct(ARRAY[1,2,2], ARRAY[2,2,4,5]) AS RESULT;
     result
  -------------
   {2}
  (1 row)

  gaussdb=# SELECT array_intersect_distinct(ARRAY[1,2,3], NULL) AS RESULT;
   result
  --------

  (1 row)
  ```

- array_except(anyarray, anyarray)

  Description: Calculates the difference between two arrays. Only one-dimensional arrays are supported. If the first input parameter is **NULL**, **NULL** is returned. If the second input parameter is **NULL**, the first input parameter is returned.

  Return type: anyarray

  Example:


```
gaussdb=# SELECT array_except(ARRAY[1,2,3], ARRAY[3,4,5]) AS RESULT;
  result
-------------
 {1,2}
(1 row)

gaussdb=# SELECT array_except(ARRAY[1,2,3], NULL) AS RESULT;
 result
---------
 {1,2,3}
(1 row)

gaussdb=# SELECT array_except(NULL, ARRAY[3,4,5]) AS RESULT;
 result
--------

(1 row)
```

- array_except_distinct(anyarray, anyarray)

  Description: Calculates the difference between two arrays and deduplicates them. Only one-dimensional arrays are supported. If the first input parameter is **NULL**, **NULL** is returned. If the second input parameter is **NULL**, the first input parameter is returned.

  Return type: anyarray

  Example:

```
gaussdb=# SELECT array_except_distinct(ARRAY[1,2,2,3], ARRAY[3,4,5]) AS RESULT;
  result
-------------
 {1,2}
(1 row)

gaussdb=# SELECT array_except_distinct(ARRAY[1,2,3], NULL) AS RESULT;
 result
---------
 {1,2,3}
(1 row)

gaussdb=# SELECT array_except_distinct(NULL, ARRAY[3,4,5]) AS RESULT;
 result
--------

(1 row)
```

- array_ndims(anyarray)

  Description: Returns the number of dimensions of an array.

  Return type: int

  Example:

```
gaussdb=# SELECT array_ndims(ARRAY[[1,2,3], [4,5,6]]) AS RESULT;
 result
--------
      2
(1 row)
```

- array_dims(anyarray)

  Description: Returns the low-order flag bits and high-order flag bits of each dimension in an array.

  Return type: text

  Example:

```
gaussdb=# SELECT array_dims(ARRAY[[1,2,3], [4,5,6]]) AS RESULT;
  result
------------
```



```
 [1:2][1:3]
(1 row)
```

- array_length(anyarray, int)

  Description: Returns the length of the requested array dimension. **int** is the requested array dimension.

  Return type: int

  Example:

```
gaussdb=# SELECT array_length(array[1,2,3], 1) AS RESULT;
 result
--------
      3
(1 row)

gaussdb=# SELECT array_length(array[[1,2,3],[4,5,6]], 2) AS RESULT;
 result
--------
      3
(1 row)
```

- array_lower(anyarray, int)

  Description: Returns lower bound of the requested array dimension. **int** is the requested array dimension.

  Return type: int

  Example:

```
gaussdb=# SELECT array_lower('[0:2]={1,2,3}'::int[], 1) AS RESULT;
 result
--------
      0
(1 row)
```

  NOTE

  If the first parameter is null, error "Reference to uninitialized collection" is reported after **varray_compat** is enabled. Before **varray_compat** is enabled, **null** is returned.

- array_sort(anyarray)

  Description: Returns an array in ascending order. Only one-dimensional array of the anyarray type supports sorting. Multidimensional arrays return **NULL**. Currently, the record, XML, XMLtype, and JSON arrays cannot be sorted.

  Return type: anyarray

  Example:

```
gaussdb=# SELECT array_sort(ARRAY[5,1,3,6,2,7]) AS RESULT;
 result
-------------
{1,2,3,5,6,7}
(1 row)
gaussdb=# SELECT array_sort(array[array[1,23], array[1,34]]);
 array_sort
-------------
 {NULL,NULL}
(1 row)
```

- array_upper(anyarray, int)

  Description: Returns upper bound of the requested array dimension. **int** is the requested array dimension.

  Return type: int

  Example:


```
gaussdb=# SELECT array_upper(ARRAY[1,8,3,7], 1) AS RESULT;
 result
--------
      4
(1 row)
```

NOTE

If the first parameter is null, error "Reference to uninitialized collection" is reported after **varray_compat** is enabled. Before **varray_compat** is enabled, **null** is returned.

- array_to_string(anyarray, text [, text])

  Description: Uses the first **text** as the new delimiter and the second **text** to replace **NULL** values.

  Return type: text

  Example:

```
gaussdb=# SELECT array_to_string(ARRAY[1, 2, 3, NULL, 5], ',', '*') AS RESULT;
  result
-----------
 1,2,3,*,5
(1 row)
```

- array_delete(anyarray)

  Description: Clears elements in an array and returns an empty array of the same type.

  Return type: anyarray

  Example:

```
gaussdb=# SELECT array_delete(ARRAY[1,8,3,7]) AS RESULT;
 result
--------
 {}
(1 row)
```

NOTE

If the first parameter is null, error "Reference to uninitialized collection" is reported after **varray_compat** is enabled. Before **varray_compat** is enabled, **null** is returned.

- array_deleteidx(anyarray, int)

  Description: Deletes specified index elements from an array and returns an array consisting of the remaining elements.

  Return type: anyarray

  Example:

```
gaussdb=# SELECT array_deleteidx(ARRAY[1,2,3,4,5], 1) AS RESULT;
  result
-----------
 {2,3,4,5}
(1 row)
```


NOTE

- **array_deleteidx(anyarray, int)**: This function is disabled when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s1**.
- After the **varray_compat** parameter is enabled, if the first parameter is null, error "Reference to uninitialized collection" is reported. If the second parameter is null, the original array is returned. If this parameter is disabled and one of the parameters is null, **null** is returned. If the value of the second parameter is less than or equal to **0**, the error "Subscript outside of limit" is reported after the parameter is enabled. Before the parameter is enabled, the original array is returned. If the value of the second parameter is greater than the number of elements in the array (including **0**, that is, an empty array), the error "Subcript outside of count" is reported after the parameter is enabled. Before the parameter is enabled, the original array is returned.

- array_extendnull(anyarray, int)

  Description: Adds a specified number of NULL elements to the end of an array.

  Return type: anyarray

  Example:

```
gaussdb=# SELECT array_extendnull(ARRAY[1,8,3,7],1) AS RESULT;
   result
--------------
{1,8,3,7,null}
(1 row)
```

  NOTE

  If the first parameter is null, error "Reference to uninitialized collection" is reported after **varray_compat** is enabled. Before **varray_compat** is enabled, **null** is returned. If the second parameter is null, **null** is returned before **varray_compat** is enabled. After **varray_compat** is enabled, the original array is returned. If the second parameter is less than **0**, error "numeric or value error" is reported after **varray_compat** is enabled. Before **varray_compat** is enabled, the original array is returned.

- array_extendnull(anyarray, int, int)

  Description: Adds a specified number of elements with a specified index to the end of an array.

  Return type: anyarray

  Example:

```
gaussdb=# SELECT array_extendnull(ARRAY[1,8,3,7],2,2) AS RESULT;
   result
--------------
{1,8,3,7,8,8}
(1 row)
```

  NOTE

  array_extendnull(anyarray, int, int): This function takes effect when the value of **a_format_version** is **10c** and the value of **a_format_dev_version** is **s1**.

  If the first parameter is null, error "Reference to uninitialized collection" is reported after **varray_compat** is enabled. Before **varray_compat** is enabled, **null** is returned. If the second or third parameter is null, **null** is returned before **varray_compat** is enabled. After **varray_compat** is enabled, the original array is returned.

- array_trim(anyarray, int)

  Description: Deletes a specified number of elements from the end of an array.

  Return type: anyarray

Example:

```
gaussdb=# SELECT array_trim(ARRAY[1,8,3,7],1) AS RESULT;
 result
---------
 {1,8,3}
(1 row)
```

NOTE

If the first parameter is null, error "Reference to uninitialized collection" is reported after **varray_compat** is enabled and **null** is returned before **varray_compat** is enabled. If the second parameter is null, **null** is returned before **varray_compat** is enabled and the original array is returned after **varray_compat** is enabled. If the second parameter exceeds the number of array elements (including **0**, that is, an empty array), error "Subscipt outside of count" is reported after **varray_compat** is enabled, and before **varray_compat** is enabled, an empty array is returned. If the value of the second parameter is less than **0**, an error message "numeric or value error" is displayed after **varray_compat** is enabled. Before **varray_compat** is enabled, the original array is returned.

- array_exists(anyarray, int)

  Description: Checks whether the second parameter is a valid index of an array.

  Return type: Boolean

  Example:

  ```
  gaussdb=# SELECT array_exists(ARRAY[1,8,3,7],1) AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- array_next(anyarray, int)

  Description: Returns the index of the element following a specified index in an array based on the second input parameter.

  Return type: int

  Example:

  ```
  gaussdb=# SELECT array_next(ARRAY[1,8,3,7],1) AS RESULT;
   result
  --------
       2
  (1 row)
  ```

  NOTE

  If the first parameter is null, error "Reference to uninitialized collection" is reported after **varray_compat** is enabled. Before **varray_compat** is enabled, **null** is returned.

- array_prior(anyarray, int)

  Description: Returns the index of the element followed by a specified index in an array based on the second input parameter.

  Return type: int

  Example:

  ```
  gaussdb=# SELECT array_prior(ARRAY[1,8,3,7],2) AS RESULT;
   result
  --------
    1
  (1 row)
  ```


> **NOTE**
>
> If the first parameter is null, error "Reference to uninitialized collection" is reported after **varray_compat** is enabled. Before **varray_compat** is enabled, **null** is returned.

- string_to_array(text, text, [text])

  Description: Uses the second **text** as the new delimiter and the third **text** as the substring to be replaced by **NULL** values. A substring can be replaced by **NULL** values only when it is the same as the third **text**.

  Return type: text[]

  Example:

```
gaussdb=# SELECT string_to_array('xx~^~yy~^~zz', '~^~', 'yy') AS RESULT;
    result
--------------
 {xx,NULL,zz}
(1 row)
gaussdb=# SELECT string_to_array('xx~^~yy~^~zz', '~^~', 'y') AS RESULT;
   result
------------
 {xx,yy,zz}
(1 row)
```

- unnest(anyarray)

  Description: Expands an array to a set of rows.

  Return type: setof anyelement

  Example:

```
gaussdb=# SELECT unnest(ARRAY[1,2]) AS RESULT;
 result
--------
      1
      2
(2 rows)
```

- unnest(anynesttable)

  Description: Returns a collection of elements in a nest-table.

  Return type: setof anyelement

  Restriction: The tableof type cannot be nested with the tableof type, or the tableof type cannot be nested with other types and then the tableof type.

  Example:

```
CREATE OR REPLACE PROCEDURE f1()
AS
    TYPE t1 IS TABLE of INT;
    v2 t1 := t1(null, 2, 3, 4, null);
    tmp INT;
    CURSOR c1 IS SELECT * FROM unnest(v2);
BEGIN
OPEN c1;
FOR i IN 1 .. v2.count LOOP
    FETCH c1 INTO tmp;
    IF tmp IS null THEN
        dbe_output.print_line(i || ': is null');
    ELSE
        dbe_output.print_line(i || ': ' || tmp);
    END IF;
END LOOP;
CLOSE c1;
END;
/

gaussdb=# CALL f1();
```

```
1: is null
2: 2
3: 3
4: 4
5: is null
 f1
----

(1 row)
```

- unnest(anyindexbytable)

  Description: Returns the collection of elements in an index-by table sorted by index.

  Return type: setof anyelement

  Restriction: The tableof type cannot be nested with the tableof type, or the tableof type cannot be nested with other types and then the tableof type. Only the index by int type is supported. The index by varchar type is not supported.

  Example:

```
CREATE OR REPLACE PROCEDURE f1()
AS
    TYPE t1 IS TABLE of INT INDEX BY INT;
    v2 t1 := t1(1=>1, -10=>(-10), 6=>6, 4=>null);
    tmp INT;
    CURSOR c1 IS SELECT * FROM unnest(v2);
BEGIN
OPEN c1;
FOR i IN 1 .. v2.count LOOP
    FETCH c1 INTO tmp;
    IF tmp IS null THEN
        dbe_output.print_line(i || ': is null');
    ELSE
        dbe_output.print_line(i || ': ' || tmp);
    END IF;
END LOOP;
CLOSE c1;
END;
/

gaussdb=# CALL f1();
1: -10
2: 1
3: is null
4: 6
 f1
----

(1 row)
```

In **string_to_array**, if the delimiter parameter is NULL, each character in the input string will become a separate element in the resulting array. If the delimiter is an empty string, then the entire input string is returned as a one-element array. Otherwise the input string is split at each occurrence of the delimiter string.

In **string_to_array**, if the null-string parameter is omitted or **NULL**, none of the substrings of the input will be replaced by **NULL**.

In **array_to_string**, if the null-string parameter is omitted or **NULL**, any null elements in the array are simply skipped and not represented in the output string.

- _pg_keysequal

  Description: Checks whether two smallint arrays are the same.

Parameter: smallint[], smallint[]

Return type: Boolean

NOTE

This function exists in the information_schema namespace.

- cardinality(anyarray)

  Description: Returns the total number of elements in each dimension of an array. If the array is empty, **0** is returned.

  Return type: integer

  Example:

```
gaussdb=# SELECT cardinality(array[[1, 2], [3, 4]]);
 cardinality
-------------
  4
(1 row)
```

- array_positions(anyarray, anyelement)

  Description: Returns an array of indexes of all second parameters that appear in the array given as the first parameter.

  Return type: int[]

  Example:

```
gaussdb=# SELECT array_positions(array[1, 2, 3, 1], 1) AS RESULT;
 result
-------------
 {1,4}
(1 row)
```

NOTE

- The array must be one-dimensional.
- The second parameter can be set to **NULL**.
- If no second parameter is found in the array, an empty array is returned.

7.6.17 Range Functions and Operators

Range Operators

- =

  Description: Equals

  Example:

```
gaussdb=# SELECT int4range(1,5) = '[1,4]'::int4range AS RESULT;
 result
-------------
 t
(1 row)
```

- <>

  Description: Does not equal

  Example:

```
gaussdb=# SELECT numrange(1.1,2.2) <> numrange(1.1,2.3) AS RESULT;
 result
-------------
 t
(1 row)
```


- <

  Description: Is less than

  Example:

```
gaussdb=# SELECT int4range(1,10) < int4range(2,3) AS RESULT;
 result
--------
 t
(1 row)
```

- >

  Description: Is greater than

  Example:

```
gaussdb=# SELECT int4range(1,10) > int4range(1,5) AS RESULT;
 result
--------
 t
(1 row)
```

- <=

  Description: Is less than or equals

  Example:

```
gaussdb=# SELECT numrange(1.1,2.2) <= numrange(1.1,2.2) AS RESULT;
 result
--------
 t
(1 row)
```

- >=

  Description: Is greater than or equals

  Example:

```
gaussdb=# SELECT numrange(1.1,2.2) >= numrange(1.1,2.0) AS RESULT;
 result
--------
 t
(1 row)
```

- @>

  Description: Contains range

  Example:

```
gaussdb=# SELECT int4range(2,4) @> int4range(2,3) AS RESULT;
 result
--------
 t
(1 row)
```

- @>

  Description: Contains element

  Example:

```
gaussdb=# SELECT '[2011-01-01,2011-03-01)'::tsrange @> '2011-01-10'::timestamp AS RESULT;
 result
--------
 t
(1 row)
```

- <@

  Description: Range is contained by

  Example:

```
gaussdb=# SELECT int4range(2,4) <@ int4range(1,7) AS RESULT;
 result
```



```
--------
 t
(1 row)
```

- <@

  Description: Element is contained by

  Example:

```
gaussdb=# SELECT 42 <@ int4range(1,7) AS RESULT;
 result
--------
 f
(1 row)
```

- &&

  Description: Overlap (have points in common)

  Example:

```
gaussdb=# SELECT int8range(3,7) && int8range(4,12) AS RESULT;
 result
--------
 t
(1 row)
```

- <<

  Description: Strictly left of

  Example:

```
gaussdb=# SELECT int8range(1,10) << int8range(100,110) AS RESULT;
 result
--------
 t
(1 row)
```

- >>

  Description: Strictly right of

  Example:

```
gaussdb=# SELECT int8range(50,60) >> int8range(20,30) AS RESULT;
 result
--------
 t
(1 row)
```

- &<

  Description: Does not extend to the right of

  Example:

```
gaussdb=# SELECT int8range(1,20) &< int8range(18,20) AS RESULT;
 result
--------
 t
(1 row)
```

- &>

  Description: Does not extend to the left of

  Example:

```
gaussdb=# SELECT int8range(7,20) &> int8range(5,10) AS RESULT;
 result
--------
 t
(1 row)
```

- -|-

  Description: Is adjacent to

Example:

```
gaussdb=# SELECT numrange(1.1,2.2) -|- numrange(2.2,3.3) AS RESULT;
 result
--------
 t
(1 row)
```

- +

  Description: Union

  Example:

  ```
  gaussdb=# SELECT numrange(5,15) + numrange(10,20) AS RESULT;
   result
  --------
   [5,20)
  (1 row)
  ```

- *

  Description: Intersection

  Example:

  ```
  gaussdb=# SELECT int8range(5,15) * int8range(10,20) AS RESULT;
   result
  ---------
   [10,15)
  (1 row)
  ```

- -

  Description: Difference

  Example:

  ```
  gaussdb=# SELECT int8range(5,15) - int8range(10,20) AS RESULT;
   result
  --------
   [5,10)
  (1 row)
  ```

The simple comparison operators **<**, **>**, **<=**, and **>=** compare the lower bounds first, and only if those are equal, compare the upper bounds.

The **<<**, **>>**, and **-|-** operators always return false when an empty range is involved; that is, an empty range is not considered to be either before or after any other range.

The union and difference operators will fail if the resulting range would need to contain two disjoint sub-ranges.

Range Functions

The lower and upper functions return null if the range is empty or the requested bound is infinite. The lower_inc, upper_inc, lower_inf, and upper_inf functions all return false for an empty range.

- numrange(numeric, numeric, [text])

  Description: Specifies a range.

  Return type: Range's element type

  Example:

  ```
  gaussdb=# SELECT numrange(1.1,2.2) AS RESULT;
   result
  --------
  [1.1,2.2)
  ```



```
(1 row)
gaussdb=# SELECT numrange(1.1,2.2, '()') AS RESULT;
 result
--------
(1.1,2.2)
(1 row)
```

- lower(anyrange)

  Description: Lower bound of a range

  Return type: Range's element type

  Example:

  ```
  gaussdb=# SELECT lower(numrange(1.1,2.2)) AS RESULT;
   result
  --------
     1.1
  (1 row)
  ```

- upper(anyrange)

  Description: Upper bound of range

  Return type: Range's element type

  Example:

  ```
  gaussdb=# SELECT upper(numrange(1.1,2.2)) AS RESULT;
   result
  --------
     2.2
  (1 row)
  ```

- isempty(anyrange)

  Description: Is the range empty?

  Return type: Boolean

  Example:

  ```
  gaussdb=# SELECT isempty(numrange(1.1,2.2)) AS RESULT;
   result
  --------
   f
  (1 row)
  ```

- lower_inc(anyrange)

  Description: Is the lower bound inclusive?

  Return type: Boolean

  Example:

  ```
  gaussdb=# SELECT lower_inc(numrange(1.1,2.2)) AS RESULT;
   result
  --------
   t
  (1 row)
  ```

- upper_inc(anyrange)

  Description: Is the upper bound inclusive?

  Return type: Boolean

  Example:

  ```
  gaussdb=# SELECT upper_inc(numrange(1.1,2.2)) AS RESULT;
   result
  --------
   f
  (1 row)
  ```

- lower_inf(anyrange)

Description: Is the lower bound infinite?

Return type: Boolean

Example:

```
gaussdb=# SELECT lower_inf('(,)'::daterange) AS RESULT;
 result
--------
 t
(1 row)
```

- upper_inf(anyrange)

  Description: Is the upper bound infinite?

  Return type: Boolean

  Example:

```
gaussdb=# SELECT upper_inf('(,)'::daterange) AS RESULT;
 result
--------
 t
(1 row)
```

- elem_contained_by_range(anyelement, anyrange)

  Description: Determines whether an element is within the range.

  Return type: Boolean

  Example:

```
gaussdb=# SELECT elem_contained_by_range('2', numrange(1.1,2.2));
 elem_contained_by_range
-------------------------
 t
(1 row)
```

7.6.18 Aggregate Functions

Aggregate Functions

- sum(expression)

  Description: Sum of expression across all input values

  Return type:

  Generally, same as the argument data type. In the following cases, type conversion occurs:

  – BIGINT for SMALLINT or INT arguments

  – NUMBER for BIGINT arguments

  – DOUBLE PRECISION for floating-point arguments

  Example:

```
gaussdb=# CREATE TABLE tab(a int);
CREATE TABLE
gaussdb=# INSERT INTO tab values(1);
INSERT 0 1
gaussdb=# INSERT INTO tab values(2);
INSERT 0 1
gaussdb=# SELECT sum(a) FROM tab;
 sum
-----
   3
(1 row)
gaussdb=# DROP TABLE tab;
```


- max(expression)

  Description: Specifies the maximum value of expression across all input values.

  Parameter type: any array, numeric, string, or date/time type

  Return type: same as the parameter type

  Example:

```
gaussdb=# CREATE TABLE max_t1(a int, b int);

gaussdb=# INSERT INTO max_t1 VALUES(1,2),(2,3),(3,4),(4,5);

gaussdb=# SELECT MAX(a) FROM max_t1;
 max
-----
   4
(1 row)

gaussdb=# DROP TABLE max_t1;
```

- min(expression)

  Description: Minimum value of expression across all input values

  Parameter type: any array, numeric, string, or date/time type

  Return type: same as the parameter type

  Example:

```
gaussdb=# CREATE TABLE min_t1(a int, b int);

gaussdb=# INSERT INTO min_t1 VALUES(1,2),(2,3),(3,4),(4,5);

gaussdb=# SELECT MIN(a) FROM min_t1;
 min
-----
   1
(1 row)

gaussdb=# DROP TABLE min_t1;
```

- avg(expression)

  Description: Average (arithmetic mean) of all input values

  Return type:

  NUMBER for any integer-type argument.

  DOUBLE PRECISION for floating-point parameters.

  otherwise the same as the argument data type.

  Example:

```
gaussdb=# CREATE TABLE avg_t1(a int, b int);

gaussdb=# INSERT INTO avg_t1 VALUES(1,2),(2,3),(3,4),(4,5);

gaussdb=# SELECT AVG(a) FROM avg_t1;
        avg
--------------------
 2.5000000000000000
(1 row)

gaussdb=# DROP TABLE avg_t1;
```

- count(expression)

  Description: Returns the number of input rows for which the value of expression is not null.

Return type: bigint

Operations on XML data are supported.

Example:

```
gaussdb=# CREATE TABLE count_t1(a int, b int);

gaussdb=# INSERT INTO count_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);

gaussdb=# SELECT COUNT(a) FROM count_t1;
 count
-------
     4
(1 row)

gaussdb=# DROP TABLE count_t1;
```

- count(*)

  Description: Returns the number of input rows.

  Return type: bigint

  Operations on XML data are supported.

  Example:

  ```
  gaussdb=# CREATE TABLE count_t1(a int, b int);

  gaussdb=# INSERT INTO count_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);

  gaussdb=# SELECT COUNT(*) FROM count_t1;
   count
  -------
       5
  (1 row)

  gaussdb=# DROP TABLE count_t1;
  ```

- median(expression) [over (query partition clause)]

  Description: Returns the median of an expression. **NULL** will be ignored by the median function during calculation. The DISTINCT keyword can be used to exclude duplicate records in an expression. The data type of the input expression can be numeric (including integer, double, and bigint) or interval. For other data types, the median cannot be calculated.

  Return type: double or interval

  Example:

  ```
  gaussdb=# SELECT median(id) FROM (values(1), (2), (3), (4), (null)) test(id);
   median
  --------
       2.5
  (1 row)
  ```

- default(column_name)

  Description: Obtains the default output value of a table column.

  Return type: text

  Example:

  ```
  -- Create a MySQL-compatible database.
  gaussdb=# CREATE DATABASE gaussdb_m WITH DBCOMPATIBILITY 'B';

  gaussdb=# \c gaussdb_m

  gaussdb_m=# CREATE TABLE t1(id int DEFAULT 100, name varchar(20) DEFAULT 'tt');

  gaussdb_m=# INSERT INTO t1 VALUES(1,'test');
  ```



```
-- Execute the query.
gaussdb_m=# SELECT default(id), default(name) FROM t1;
 default | default
---------+---------
     100 | tt
(1 row)

-- Delete the database.
gaussdb_m=# \c postgres
gaussdb=# DROP DATABASE gaussdb_m;
```

NOTE

- The default function takes effect only when **sql_compatibility** is set to **'B'**.
- If a table column does not have a default value, the default function returns a null value.
- If a table column is a hidden column (such as **xmin** or **cmin**), the default function returns a null value.
- If a table column is an auto-increment column, the default function returns **0**.
- GaussDB supports default values of partitioned tables, temporary tables, and multi-table join query.
- GaussDB supports the query of nodes whose column names contain character string values (indicating names) and A_Star nodes (indicating that asterisks (*) appear), for example, **default(tt.t4.id)** and **default(tt.t4.*)**. For invalid query column names and A_Star nodes, the error information reported by GaussDB is different from that reported by B-compatible database.
- When the default value of a column is created in GaussDB, the range of the column type is not verified. As a result, an error may be reported when the default function is used.
- If the default value of a column is a function expression, the default function in GaussDB returns the calculated value of the default expression of the column during table creation. The default function of B-compatible database returns **NULL**.

- array_agg(expression)

  Description: Input values, including nulls, concatenated into an array

  Return type: array of the parameter type

  Operations on XML data are supported.

  Example:

```
gaussdb=# CREATE TABLE array_agg_t1(a int, b int);

gaussdb=# INSERT INTO array_agg_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);

gaussdb=# SELECT ARRAY_AGG(a) FROM array_agg_t1;
  array_agg
---------------
 {NULL,1,2,3,4}
(1 row)

gaussdb=# DROP TABLE array_agg_t1;
```

- string_agg(expression, delimiter)

  Description: Input values concatenated into a string, separated by delimiter

  Return type: same as the parameter data type.

  Operations on XML data that is explicitly converted to the character type are supported.

  Example:

```
gaussdb=# CREATE TABLE string_agg_t1(a int, b int);
```



```
gaussdb=# INSERT INTO string_agg_t1 VALUES (NULL,1),(1,2),(2,3),(3,4),(4,5);

gaussdb=# SELECT STRING_AGG(a,';') FROM string_agg_t1;
 string_agg
------------
 1;2;3;4
(1 row)

gaussdb=# DROP TABLE string_agg_t1;
```

- listagg(expression [, delimiter]) WITHIN GROUP(ORDER BY order-list)

  Description: Aggregation column data sorted according to the mode specified by WITHIN GROUP, and concatenated to a string using the specified delimiter

  – **expression**: Required. It specifies an aggregation column name or a column-based, valid expression. It does not support the DISTINCT keyword and the parameters of VARIADIC.

  – **delimiter**: Optional. It specifies a delimiter, which can be a string constant or a deterministic expression based on a group of columns. The default value is empty.

  – **order-list**: Required. It specifies the sorting mode in a group.

  Return type: text

  Example:

  The aggregation column is of the text character set type.

```
gaussdb=# CREATE TABLE listagg_t1(a int, b text);

gaussdb=# INSERT INTO listagg_t1 VALUES (NULL,'a1'),(1,'b2'),(1,'c3'),(2,'d4'),(2,'e5'),(3,'f6');

gaussdb=# SELECT a,LISTAGG(b,';') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
 a | listagg
---+---------
 1 | b2;c3
 2 | d4;e5
 3 | f6
   | a1
(4 rows)

gaussdb=# DROP TABLE listagg_t1;
```

  The aggregation column is of the integer type.

```
gaussdb=# CREATE TABLE listagg_t1(a int, b int);

gaussdb=# INSERT INTO listagg_t1 VALUES (NULL,1),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT a,LISTAGG(b,';') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
 a | listagg
---+---------
 1 | 2;3
 2 | 4;5
 3 | 6
   | 1
(4 rows)

gaussdb=# DROP TABLE listagg_t1;
```

  The aggregation column is of the floating point type.

```
gaussdb=# CREATE TABLE listagg_t1(a int, b float);

gaussdb=# INSERT INTO listagg_t1 VALUES (NULL,1.111),(1,2.222),(1,3.333),(2,4.444),(2,5.555),
(3,6.666);

gaussdb=# SELECT a,LISTAGG(b,';') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
 a |      listagg
---+-------------------
```



```
 1 | 2.222000;3.333000
 2 | 4.444000;5.555000
 3 | 6.666000
   | 1.111000
(4 rows)

gaussdb=# DROP TABLE listagg_t1;
```

The aggregation column is of the time type.

```
gaussdb=# CREATE TABLE listagg_t1(a int, b timestamp);

gaussdb=# INSERT INTO listagg_t1 VALUES (NULL,'2000-01-01'),(1,'2000-02-02'),(1,'2000-03-03'),
(2,'2000-04-04'),(2,'2000-05-05'),(3,'2000-06-06');

gaussdb=# SELECT a,LISTAGG(b,';') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
 a |                 listagg
---+-----------------------------------------
 1 | 2000-02-02 00:00:00;2000-03-03 00:00:00
 2 | 2000-04-04 00:00:00;2000-05-05 00:00:00
 3 | 2000-06-06 00:00:00
   | 2000-01-01 00:00:00
(4 rows)

gaussdb=# DROP TABLE listagg_t1;
```

The aggregation column is of the time interval type.

```
gaussdb=# CREATE TABLE listagg_t1(a int, b interval);

gaussdb=# INSERT INTO listagg_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'3 days'),(2,'4 days'),(2,'5
days'),(3,'6 days');

gaussdb=# SELECT a,LISTAGG(b,';') WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
 a |    listagg
---+---------------
 1 | 2 days;3 days
 2 | 4 days;5 days
 3 | 6 days
   | 1 day
(4 rows)

gaussdb=# DROP TABLE listagg_t1;
```

By default, the delimiter is empty.

```
gaussdb=# CREATE TABLE listagg_t1(a int, b interval);

gaussdb=# INSERT INTO listagg_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'3 days'),(2,'4 days'),(2,'5
days'),(3,'6 days');

gaussdb=# SELECT a,LISTAGG(b) WITHIN GROUP(ORDER BY b) FROM listagg_t1 group by a;
 a |   listagg
---+---------------
 1 | 2 days3 days
 2 | 4 days5 days
 3 | 6 days
   | 1 day
(4 rows)

gaussdb=# DROP TABLE listagg_t1;
```

When listagg is used as a window function, the OVER clause does not support the window sorting of ORDER BY, and the listagg column is an ordered aggregation of the corresponding groups.

```
gaussdb=# CREATE TABLE listagg_t1(a int, b interval);

gaussdb=# INSERT INTO listagg_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'3 days'),(2,'4 days'),(2,'5
days'),(3,'6 days');

gaussdb=# SELECT a,LISTAGG(b) WITHIN GROUP(ORDER BY b) OVER(PARTITION BY a) FROM
listagg_t1;
```



```
 a |   listagg
---+--------------
 1 | 2 days3 days
 1 | 2 days3 days
 2 | 4 days5 days
 2 | 4 days5 days
 3 | 6 days
   | 1 day
(6 rows)

gaussdb=# DROP TABLE listagg_t1;
```

- group_concat([DISTINCT | ALL] expression [,expression ...] [ORDER BY { expression [[ASC | DESC | USING operator] | nlssort_expression_clause] [NULLS { FIRST | LAST }] } [,...]] [SEPARATOR str_val])

  Description: The number of parameters is not fixed. Multiple columns can be concatenated. Aggregation columns are sorted based on the value of **ORDER BY** and concatenated into a character string using the specified separator. This function cannot be used as a window function.

  - **DISTINCT**: Optional. It deduplicates the results after each row is concatenated.
  - **expression**: Required. It specifies the aggregation column name or a valid column-based expression.
  - **ORDER BY**: Optional. It is followed by a variable number of expressions and sorting rule. The group_concat function does not support the (ORDER BY + number) format.
  - **SEPARATOR**: Optional. It is followed by a character or string. This separator is used to concatenate the expression results of two adjacent lines in a group. If it is not specified, the comma (,) is used by default.
  - When both DISTINCT and ORDER BY are specified, all ORDER BY expressions must be in DISTINCT expressions. Otherwise, an error is reported.

  Return type: text

  Example:

  Set **separator** to **';'**.

  ```
  gaussdb=# CREATE TABLE group_concat_t1(a int, b int);

  gaussdb=# INSERT INTO group_concat_t1 VALUES (NULL,1),(1,2),(1,3),(2,4),(2,5),(3,6);

  gaussdb=# SELECT a,group_concat(b separator ';') FROM group_concat_t1 GROUP BY a ORDER BY a;
   a | group_concat
  ---+--------------
   1 | 2;3
   2 | 4;5
   3 | 6
     | 1
  (4 rows)

  gaussdb=# DROP TABLE group_concat_t1;
  ```

  By default, the separator is a comma (,).

  ```
  gaussdb=# CREATE TABLE group_concat_t1(a int, b int);

  gaussdb=# INSERT INTO group_concat_t1 VALUES (NULL,1),(1,2),(1,3),(2,4),(2,5),(3,6);

  gaussdb=# SELECT a,group_concat(a,b) FROM group_concat_t1 GROUP BY a ORDER BY a;
   a | group_concat
  ---+--------------
   1 | 12,13
  ```



```
 2 | 24,25
 3 | 36
   | 1
(4 rows)

gaussdb=# DROP TABLE group_concat_t1;
```

The aggregation column is of the text character set type.

```
gaussdb=# CREATE TABLE group_concat_t1(a int, b text);

gaussdb=# INSERT INTO group_concat_t1 VALUES (NULL,'a1'),(1,'b2'),(1,'c3'),(2,'d4'),(2,'e5'),(3,'f6');

gaussdb=# SELECT a,group_concat(a,b) FROM group_concat_t1 GROUP BY a ORDER BY a;
 a | group_concat
---+--------------
 1 | 1b2,1c3
 2 | 2d4,2e5
 3 | 3f6
   | a1
(4 rows)

gaussdb=# DROP TABLE group_concat_t1;
```

The aggregation column is of the integer type.

```
gaussdb=# CREATE TABLE group_concat_t1(a int, b int);

gaussdb=# INSERT INTO group_concat_t1 VALUES (NULL,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,group_concat(b) FROM group_concat_t1 GROUP BY a ORDER BY a;
 a | group_concat
---+--------------
 1 | 2,3
 2 | 4,5
 3 | 6
   | 1
(4 rows)

gaussdb=# DROP TABLE group_concat_t1;
```

The aggregation column is of the floating point type.

```
gaussdb=# CREATE TABLE group_concat_t1(a int, b float);

gaussdb=# INSERT INTO group_concat_t1 VALUES (NULL,1.11),(1,2.22),(1,3.33),(2,4.44),(2,5.55),
(3,6.66);

gaussdb=# SELECT a,group_concat(b) FROM group_concat_t1 GROUP BY a ORDER BY a;
 a | group_concat
---+--------------
 1 | 2.22,3.33
 2 | 4.44,5.55
 3 | 6.66
   | 1.11
(4 rows)

gaussdb=# DROP TABLE group_concat_t1;
```

The aggregation column is of the time type.

```
gaussdb=# CREATE TABLE group_concat_t1(a int, b timestamp);

gaussdb=# INSERT INTO group_concat_t1 VALUES (NULL,'2000-01-01'),(1,'2000-02-02'),
(1,'2000-03-03'),(2,'2000-04-04'),(2,'2000-05-05'),(3,'2000-06-06');

gaussdb=# SELECT a,group_concat(b) FROM group_concat_t1 GROUP BY a ORDER BY a;
 a |              group_concat
---+-----------------------------------------
 1 | 2000-02-02 00:00:00,2000-03-03 00:00:00
 2 | 2000-04-04 00:00:00,2000-05-05 00:00:00
 3 | 2000-06-06 00:00:00
   | 2000-01-01 00:00:00
```



```
(4 rows)

gaussdb=# DROP TABLE group_concat_t1;
```

The aggregation column is of the binary type.

```
gaussdb=# CREATE TABLE group_concat_t1(a int, b bytea);

gaussdb=# INSERT INTO group_concat_t1 VALUES (NULL,'1'),(1,'2'),(1,'3'),(2,'4'),(2,'5'),(3,'6');

gaussdb=# SELECT a,group_concat(b) FROM group_concat_t1 GROUP BY a ORDER BY a;
 a | group_concat
---+--------------
 1 | \x32,\x33
 2 | \x34,\x35
 3 | \x36
   | \x31
(4 rows)

gaussdb=# DROP TABLE group_concat_t1;
```

The aggregation column is of the time interval type.

```
gaussdb=# CREATE TABLE group_concat_t1(a int, b interval);

gaussdb=# INSERT INTO group_concat_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'3 days'),(2,'4 days'),
(2,'5 days'),(3,'6 days');

gaussdb=# SELECT a,group_concat(b) FROM group_concat_t1 GROUP BY a ORDER BY a;
 a | group_concat
---+---------------
 1 | 2 days,3 days
 2 | 4 days,5 days
 3 | 6 days
   | 1 day
(4 rows)

gaussdb=# DROP TABLE group_concat_t1;
```

Set **distinct** to deduplicate data.

```
gaussdb=# CREATE TABLE group_concat_t1(a int, b interval);

gaussdb=# INSERT INTO group_concat_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'2 days'),(1,'3 days'),
(1,'3 days'),(2,'4 days'),(2,'5 days'),(3,'6 days');

gaussdb=# SELECT a,group_concat(distinct b) FROM group_concat_t1 GROUP BY a ORDER BY a;
 a | group_concat
---+---------------
 1 | 2 days,3 days
 2 | 4 days,5 days
 3 | 6 days
   | 1 day
(4 rows)

gaussdb=# DROP TABLE group_concat_t1;
```

Set **ORDER BY** to sort data.

```
gaussdb=# CREATE TABLE group_concat_t1(a int, b interval);

gaussdb=# INSERT INTO group_concat_t1 VALUES (NULL,'1 days'),(1,'2 days'),(1,'3 days'),(2,'4 days'),
(2,'5 days'),(3,'6 days');

gaussdb=# SELECT a,group_concat(b ORDER BY b desc) FROM group_concat_t1 GROUP BY a ORDER
BY a;
 a | group_concat
---+---------------
 1 | 3 days,2 days
 2 | 5 days,4 days
 3 | 6 days
   | 1 day
(4 rows)
```



```
gaussdb=# DROP TABLE group_concat_t1;
```

- wm_concat(expression)

  Description: Concatenates column data into a string separated by commas (,).

  Return type: same as the parameter data type.

  NOTE

  wm_concat is developed for compatibility with database A. Currently, this function has been canceled and replaced by the listagg function in the latest version of database A. You can also use the string_agg function. For details, see the description of the two functions.

- covar_pop(Y, X)

  Description: Overall covariance

  Return type: double precision

  Example:

```
gaussdb=# CREATE TABLE covar_pop_t1(a int, b int);

gaussdb=# INSERT INTO covar_pop_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);

gaussdb=# SELECT COVAR_POP(a,b) FROM covar_pop_t1;
 covar_pop
-----------
       100
(1 row)

gaussdb=# DROP TABLE covar_pop_t1;
```

- covar_samp(Y, X)

  Description: Sample covariance

  Return type: double precision

  Example:

```
gaussdb=# CREATE TABLE covar_samp_t1(a int, b int);

gaussdb=# INSERT INTO covar_samp_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);

gaussdb=# SELECT COVAR_SAMP(a,b) FROM covar_samp_t1;
 covar_samp
------------
        125
(1 row)

gaussdb=# DROP TABLE covar_samp_t1;
```

- stddev_pop(expression)

  Description: Overall standard difference

  Return type: DOUBLE PRECISION for floating-point arguments, otherwise NUMERIC

  Example:

```
gaussdb=# CREATE TABLE stddev_pop_t1(a int, b int);

gaussdb=# INSERT INTO stddev_pop_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);

gaussdb=# SELECT STDDEV_POP(a) FROM stddev_pop_t1;
    stddev_pop
--------------------
 7.4833147735478828
(1 row)
```



```
gaussdb=# DROP TABLE stddev_pop_t1;
```

- stddev_samp(expression)

  Description: Sample standard deviation of the input values

  Return type: DOUBLE PRECISION for floating-point arguments, otherwise NUMERIC

  Example:

```
gaussdb=# CREATE TABLE stddev_samp_t1(a int, b int);

gaussdb=# INSERT INTO stddev_samp_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);

gaussdb=# SELECT STDDEV_SAMP(a) FROM stddev_samp_t1;
   stddev_samp
--------------------
 8.3666002653407555
(1 row)

gaussdb=# DROP TABLE stddev_samp_t1;
```

- var_pop(expression)

  Description: Specifies the population variance of the input values (square of the population standard deviation).

  Return type: DOUBLE PRECISION for floating-point arguments, otherwise NUMERIC

  Example:

```
gaussdb=# CREATE TABLE var_pop_t1(a int, b int);

gaussdb=# INSERT INTO var_pop_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);

gaussdb=# SELECT VAR_POP(a) FROM var_pop_t1;
      var_pop
--------------------
 56.0000000000000000
(1 row)

gaussdb=# DROP TABLE var_pop_t1;
```

- var_samp(expression)

  Description: Specifies the sample variance of the input values (square of the sample standard deviation).

  Return type: DOUBLE PRECISION for floating-point arguments, otherwise NUMERIC

  Example:

```
gaussdb=# CREATE TABLE var_samp_t1(a int, b int);

gaussdb=# INSERT INTO var_samp_t1 VALUES (NULL,11),(11,21),(11,31),(21,41),(21,51),(31,61);

gaussdb=# SELECT VAR_SAMP(a) FROM var_samp_t1;
     var_samp
--------------------
 70.0000000000000000
(1 row)

gaussdb=# DROP TABLE var_samp_t1;
```

- bit_and(expression)

  Description: The bitwise AND of all non-null input values, or null if none

  Return type: same as the parameter data type.

  Example:


```
gaussdb=# CREATE TABLE bit_and_t1(a int, b int);

gaussdb=# INSERT INTO bit_and_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT BIT_AND(a) FROM bit_and_t1;
 bit_and
---------
       0
(1 row)

gaussdb=# DROP TABLE bit_and_t1;
```

- bit_or(expression)

  Description: The bitwise OR of all non-null input values, or null if none

  Return type: same as the parameter data type.

  Example:

```
gaussdb=# CREATE TABLE bit_or_t1(a int, b int);

gaussdb=# INSERT INTO bit_or_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT BIT_OR(a) FROM bit_or_t1;
 bit_or
--------
      3
(1 row)

gaussdb=# DROP TABLE bit_or_t1;
```

- bool_and(expression)

  Description: Its value is **true** if all input values are **true**, otherwise **false**.

  Return type: Boolean

  Example:

```
gaussdb=# SELECT bool_and(100 <2500);
 bool_and
----------
 t
(1 row)
```

- bool_or(expression)

  Description: Its value is **true** if at least one input value is **true**, otherwise **false**.

  Return type: Boolean

  Example:

```
gaussdb=# SELECT bool_or(100 <2500);
 bool_or
----------
 t
(1 row)
```

- corr(Y, X)

  Description: Specifies the correlation coefficient.

  Return type: double precision

  Example:

```
gaussdb=# CREATE TABLE corr_t1(a int, b int);

gaussdb=# INSERT INTO corr_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT CORR(a,b) FROM corr_t1;
       corr
------------------
```



```
 .944911182523068
(1 row)

gaussdb=# DROP TABLE corr_t1;
```

- every(expression)

  Description: Equivalent to **bool_and**

  Return type: Boolean

  Example:

  ```
  gaussdb=# SELECT every(100 <2500);
   every
  -------
   t
  (1 row)
  ```

- regr_avgx(Y, X)

  Description: Specifies the average of the independent variable (**sum(X)/Y**).

  Return type: double precision

  Example:

  ```
  gaussdb=# CREATE TABLE regr_t1(a int, b int);

  gaussdb=# INSERT INTO regr_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

  gaussdb=# SELECT REGR_AVGX(a,b) FROM regr_t1;
   regr_avgx
  -----------
           4
  (1 row)

  gaussdb=# DROP TABLE regr_t1;
  ```

- regr_avgy(Y, X)

  Description: Specifies the average of the dependent variable (**sum(Y)/X**).

  Return type: double precision

  Example:

  ```
  gaussdb=# CREATE TABLE regr_avgy_t1(a int, b int);

  gaussdb=# INSERT INTO regr_avgy_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

  gaussdb=# SELECT REGR_AVGY(a,b) FROM regr_avgy_t1;
   regr_avgy
  -----------
         1.8
  (1 row)

  gaussdb=# DROP TABLE regr_avgy_t1;
  ```

- regr_count(Y, X)

  Description: Specifies the number of input rows in which both expressions are non-null.

  Return type: bigint

  Example:

  ```
  gaussdb=# CREATE TABLE regr_count_t1(a int, b int);

  gaussdb=# INSERT INTO regr_count_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

  gaussdb=# SELECT REGR_COUNT(a,b) FROM regr_count_t1;
   regr_count
  ------------
            5
  ```



```
(1 row)

gaussdb=# DROP TABLE regr_count_t1;
```

- regr_intercept(Y, X)

  Description: y-intercept of the least-squares-fit linear equation determined by the (X, Y) pairs

  Return type: double precision

  Example:

```
gaussdb=# CREATE TABLE regr_intercept_t1(a int, b int);

gaussdb=# INSERT INTO regr_intercept_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT REGR_INTERCEPT(b,a) FROM regr_intercept_t1;
  regr_intercept
------------------
 .785714285714286
(1 row)

gaussdb=# DROP TABLE regr_intercept_t1;
```

- regr_r2(Y, X)

  Description: Specifies the square of the correlation coefficient.

  Return type: double precision

  Example:

```
gaussdb=# CREATE TABLE regr_r2_t1(a int, b int);

gaussdb=# INSERT INTO regr_r2_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT REGR_R2(b,a) FROM regr_r2_t1;
     regr_r2
------------------
 .892857142857143
(1 row)

gaussdb=# DROP TABLE regr_r2_t1;
```

- regr_slope(Y, X)

  Description: Slope of the least-squares-fit linear equation determined by the (X, Y) pairs

  Return type: double precision

  Example:

```
gaussdb=# CREATE TABLE regr_slope_t1(a int, b int);

gaussdb=# INSERT INTO regr_slope_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT REGR_SLOPE(b,a) FROM regr_slope_t1;
    regr_slope
------------------
 1.78571428571429
(1 row)

gaussdb=# DROP TABLE regr_slope_t1;
```

- regr_sxx(Y, X)

  Description: **sum(Y^2) – sum(X)^2/N** (sum of squares of the independent variables)

  Return type: double precision

  Example:

```
gaussdb=# CREATE TABLE regr_sxx_t1(a int, b int);

gaussdb=# INSERT INTO regr_sxx_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT REGR_SXX(b,a) FROM regr_sxx_t1;
 regr_sxx
----------
      2.8
(1 row)

gaussdb=# DROP TABLE regr_sxx_t1;
```

- regr_sxy(Y, X)

  Description: **sum(X*Y) - sum(X) * sum(Y)/N** (sum of products of independent times dependent variable)

  Return type: double precision

  Example:

```
gaussdb=# CREATE TABLE regr_sxy_t1(a int, b int);

gaussdb=# INSERT INTO regr_sxy_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT REGR_SXY(b,a) FROM regr_sxy_t1;
 regr_sxy
----------
        5
(1 row)

gaussdb=# DROP TABLE regr_sxy_t1;
```

- regr_syy(Y, X)

  Description: **sum(Y^2) - sum(X)^2/N** (sum of squares of the dependent variable)

  Return type: double precision

  Example:

```
gaussdb=# CREATE TABLE regr_syy_t1(a int, b int);

gaussdb=# INSERT INTO regr_syy_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT REGR_SYY(b,a) FROM regr_syy_t1;
 regr_syy
----------
       10
(1 row)

gaussdb=# DROP TABLE regr_syy_t1;
```

- stddev(expression)

  Description: Alias of **stddev_samp**

  Return type: DOUBLE PRECISION for floating-point arguments, otherwise NUMERIC

  Example:

```
gaussdb=# CREATE TABLE stddev_t1(a int, b int);

gaussdb=# INSERT INTO stddev_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT STDDEV(a) FROM stddev_t1;
       stddev
----------------------
 .83666002653407554798
(1 row)

gaussdb=# DROP TABLE stddev_t1;
```


- variance(expexpression,ression)

  Description: Alias of **var_samp**

  Return type: DOUBLE PRECISION for floating-point arguments, otherwise NUMERIC

  Example:

```
gaussdb=# CREATE TABLE variance_t1(a int, b int);

gaussdb=# INSERT INTO variance_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT VARIANCE(a) FROM variance_t1;
      variance
----------------------
 .70000000000000000000
(1 row)

gaussdb=# DROP TABLE variance_t1;
```

- delta

  Description: Returns the difference between the current row and the previous row.

  Parameter: numeric

  Return type: numeric

- checksum(expression)

  Description: Returns the CHECKSUM value of all input values. This function can be used to check whether the data in the tables is the same before and after the backup, restoration, or migration of GaussDB (databases other than GaussDB are not supported). Before and after database backup, database restoration, or data migration, you need to manually run SQL commands to obtain the execution results. Compare the obtained execution results to check whether the data in the tables before and after the backup or migration is the same.

  NOTE

  - For large tables, the execution of CHECKSUM function may take a long time.
  - If the CHECKSUM values of two tables are different, it indicates that the contents of the two tables are different. Using the hash function in the CHECKSUM function may incur conflicts. There is low possibility that two tables with different contents may have the same CHECKSUM value. The same problem may occur when CHECKSUM is used for columns.
  - If the time type is timestamp, timestamptz, or smalldatetime, ensure that the time zone settings are the same when calculating the CHECKSUM value.

  – If the CHECKSUM value of a column is calculated and the column type can be changed to TEXT by default, set *expression* to the column name.
  – If the CHECKSUM value of a column is calculated and the column type cannot be converted to TEXT by default, set *expression* to *Column name*::**TEXT**.
  – If the CHECKSUM value of all columns is calculated, set *expression* to *Table name*::**TEXT**.

  The following types of data can be converted into the TEXT type by default: char, name, int8, int2, int1, int4, raw, pg_node_tree, float4, float8, bpchar, varchar, nvarchar, nvarchar2, date, timestamp, timestamptz, numeric, and smalldatetime. Other types need to be forcibly converted to TEXT.

Return type: numeric

Example:

The following shows the CHECKSUM value of a column that can be converted to the TEXT type by default:

```
gaussdb=# CREATE TABLE checksum_t1(a int, b int);

gaussdb=# INSERT INTO checksum_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT CHECKSUM(a) FROM checksum_t1;
  checksum
-------------
 18126842830
(1 row)

gaussdb=# DROP TABLE checksum_t1;
```

The following shows the CHECKSUM value of a column that cannot be converted to the TEXT type by default. Note that the CHECKSUM parameter is set to *Column name*::**TEXT**.

```
gaussdb=# CREATE TABLE checksum_t1(a int, b int);

gaussdb=# INSERT INTO checksum_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT CHECKSUM(a::TEXT) FROM checksum_t1;
  checksum
-------------
 18126842830
(1 row)

gaussdb=# DROP TABLE checksum_t1;
```

The following shows the CHECKSUM value of all columns in a table. Note that the CHECKSUM parameter is set to *Table name*::**TEXT**. The table name is not modified by its schema.

```
gaussdb=# CREATE TABLE checksum_t1(a int, b int);

gaussdb=# INSERT INTO checksum_t1 VALUES (NULL,11),(1,2),(1,3),(2,4),(2,5),(3,6);

gaussdb=# SELECT CHECKSUM(checksum_t1::TEXT) FROM checksum_t1;
  checksum
-------------
 11160522226
(1 row)

gaussdb=# DROP TABLE checksum_t1;
```

- percentile_cont(percentile float)

  Description: Sorts a given column by time series and returns a percentile value.

  Return type: float

  NOTE

  - **percentile** is a decimal from 0 to 1. The value is of the floating point type and does not support the percent sign (%), for example, 95%.
  - This parameter must be used together with WITHIN GROUP (ORDER BY) to specify the column to be calculated. The column must be of the numeric type.

  Example:

```
gaussdb=# SELECT percentile_cont(0) WITHIN GROUP (ORDER BY value) FROM (VALUES (1),(2))
v(value);
 percentile_cont
-----------------
```



```
          1
(1 row)
```

- mode() WITHIN GROUP (ORDER BY value anyelement)

  Description: Returns the value with the highest occurrence frequency in a column. If multiple values have the same frequency, the smallest value is returned. The sorting mode is the same as the default sorting mode of the column type. **value** is an input parameter and can be of any type.

  Return type: same as the input parameter type

  Example:

```
gaussdb=# SELECT mode() WITHIN GROUP (ORDER BY value) FROM (values(1, 'a'), (2, 'b'), (2, 'c'))
v(value, tag);
 mode
------
   2
(1 row)
gaussdb=# SELECT mode() WITHIN GROUP (ORDER BY tag) FROM (values(1, 'a'), (2, 'b'), (2, 'c'))
v(value, tag);
 mode
------
 a
(1 row)
```

- pivot_func(anyelement)

  Description: Returns the only non-null value in a column. If there are two or more non-null values, an error is reported. **value** is an input parameter and can be of any type.

  Return type: same as the input parameter type

  > NOTE
  >
  > This aggregate function is mainly used inside the pivot syntax.

  Example:

```
gaussdb=# CREATE TABLE pivot_func_t1(a int, b int);

gaussdb=# INSERT INTO pivot_func_t1 VALUES (NULL,11),(1,2);

gaussdb=# SELECT PIVOT_FUNC(a) FROM pivot_func_t1;
 pivot_func
------------
          1
(1 row)

gaussdb=# DROP TABLE pivot_func_t1;
```

Aggregate Function Nesting

Description: Performs another aggregate function operation on the grouping calculation result of the aggregate function.

Generally, it can be described as follows:

```
SELECT AGG1(AGG2(column_name1)) FROM table_name GROUP BY column_name2;
```

It is equivalent to:

```
SELECT AGG1(value) FROM (SELECT AGG2(column_name1) value FROM table_name GROUP BY
column_name2);
```

In the preceding information:

- **AGG1()**: outer aggregate function.
- **AGG2()**: inner aggregate function.
- **table_name**: table name.
- **column_name1** and **column_name2**: column names.
- **value**: alias of the result of the inner aggregate function.

The overall meaning can be described as follows: The grouping calculation result of the inner aggregate function AGG2() is used as the input of the outer aggregate function AGG1() for recalculation.

NOTE

1. The nested aggregate function is located between SELECT and FROM. Otherwise, it is meaningless.
2. The SELECT statement that uses a nested aggregate function contains a GROUP BY clause.
3. Only the nested aggregate functions or constant expressions can be selected together with another nested aggregate function.
4. The aggregate function supports only one nesting operation.
5. Currently, the following aggregate functions can be nested: avg, max, min, sum, var_pop, var_samp, variance, stddev_pop, stddev_samp, stddev, median, regr_sxx, regr_syy, regr_sxy, regr_avgx, regr_avgy, regr_r2, regr_slope, regr_intercept, covar_pop, covar_samp, corr, and listagg.
6. The return type of the inner aggregate function must comply with the parameter type of the outer aggregate function.

Example:

```
gaussdb=# CREATE TABLE test1 (id INT,val INT);
CREATE TABLE
gaussdb=# INSERT INTO test1 VALUES (1, 1);
INSERT 0 1
gaussdb=# INSERT INTO test1 VALUES (1, null);
INSERT 0 1
gaussdb=# INSERT INTO test1 VALUES (2, 10);
INSERT 0 1
gaussdb=# INSERT INTO test1 VALUES (2, 55);
INSERT 0 1

gaussdb=# SELECT SUM(MIN(val)) FROM test1 GROUP BY id;
 sum
-----
  11
(1 row)

gaussdb=# DROP TABLE test1;
DROP TABLE
```

7.6.19 Window Functions

Window Functions

This statement is used together with the window function. The OVER clause is used for grouping data and sorting the elements in a group. Window functions are used for generating sequence numbers for the values in the group.

> **NOTE**
>
> **order by** in a window function must be followed by a column name. If it is followed by a number, the number is processed as a constant value and the target column is not ranked.

If a parent query contains filter criteria of a window function in a subquery, the filter criteria in the parent query can be pushed down to the subquery.

> **CAUTION**
>
> 1. Only the <, <=, and = filter criteria of window functions in parent queries can be pushed down to subqueries.
> 2. The upper limit can be a constant, constant expression, parameter, non-VOLATILE function, or non-correlated sublink.
> 3. Only the **ROW_NUMBER()**, **RANK()**, and **DENSE_RANK()** window functions are supported.

Example:

```
-- Create a table.
gaussdb=# CREATE TABLE t2 (c1 INT, c2 INT);
-- Import data.
gaussdb=# INSERT INTO t2 SELECT generate_series, generate_series FROM generate_series(1, 1000000);
-- Execute the query. The query is normal, no error is reported, and the execution result is correct.
gaussdb=# EXPLAIN ANALYZE SELECT nc1 FROM (
SELECT row_number() over() rid,
t2.c1 nc1
FROM t2
) WHERE rid BETWEEN 1 AND (1 + 10 - 1);
                                                  QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Subquery Scan on __unnamed_subquery__  (cost=0.00..0.42 rows=3 width=4) (actual time=0.201..0.228
rows=10 loops=1)
   Filter: (__unnamed_subquery__.rid >= 1)
   ->  WindowAgg  (cost=0.00..0.30 rows=10 width=4) (actual time=0.191..0.211 rows=10 loops=1)
         row_number_filter: (row_number() OVER () <= 10)
         ->  Seq Scan on t2  (cost=0.00..11977.45 rows=817445 width=4) (actual time=0.150..0.153 rows=11
loops=1)
 Total runtime: 0.539 ms
(6 rows)
-- Clear the environment to prevent data leakage.
gaussdb=# DROP TABLE t2;
```

- RANK()

  Description: The RANK function is used for generating non-consecutive sequence numbers for the values in each group. The same values have the same sequence number.

  Return type: bigint

  Example:

  ```
  gaussdb=# CREATE TABLE rank_t1(a int, b int);

  gaussdb=# INSERT INTO rank_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

  gaussdb=# SELECT a,b,RANK() OVER(PARTITION BY a ORDER BY b) FROM rank_t1;
   a | b | rank
  ---+---+------
   1 | 1 |    1
   1 | 1 |    1
   1 | 2 |    3
   1 | 3 |    4
  ```



```
 2 | 4 |    1
 2 | 5 |    2
 3 | 6 |    1
(7 rows)

gaussdb=# DROP TABLE rank_t1;
```

- ROW_NUMBER()

  Description: The ROW_NUMBER function is used for generating consecutive sequence numbers for the values in each group. The same values have different sequence numbers.

  Return type: bigint

  Example:

```
gaussdb=# CREATE TABLE row_number_t1(a int, b int);

gaussdb=# INSERT INTO row_number_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,ROW_NUMBER() OVER(PARTITION BY a ORDER BY b) FROM row_number_t1;
 a | b | row_number
---+---+------------
 1 | 1 |          1
 1 | 1 |          2
 1 | 2 |          3
 1 | 3 |          4
 2 | 4 |          1
 2 | 5 |          2
 3 | 6 |          1
(7 rows)

gaussdb=# DROP TABLE row_number_t1;
```

- DENSE_RANK()

  Description: The DENSE_RANK function is used for generating consecutive sequence numbers for the values in each group. The same values have the same sequence number.

  Return type: bigint

  Example:

```
gaussdb=# CREATE TABLE dense_rank_t1(a int, b int);

gaussdb=# INSERT INTO dense_rank_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,DENSE_RANK() OVER(PARTITION BY a ORDER BY b) FROM dense_rank_t1;
 a | b | dense_rank
---+---+------------
 1 | 1 |          1
 1 | 1 |          1
 1 | 2 |          2
 1 | 3 |          3
 2 | 4 |          1
 2 | 5 |          2
 3 | 6 |          1
(7 rows)

gaussdb=# DROP TABLE dense_rank_t1;
```

- PERCENT_RANK()

  Description: The PERCENT_RANK function is used for generating corresponding sequence numbers for the values in each group. That is, the function calculates the value according to the formula: Sequence number = (**rank** - 1) / (**totalrows** - 1). **rank** is the corresponding sequence number generated based on the **RANK** function for the value and **totalrows** is the total number of elements in a group.

Return type: double precision

Example:

```
gaussdb=# CREATE TABLE percent_rank_t1(a int, b int);

gaussdb=# INSERT INTO percent_rank_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,PERCENT_RANK() OVER(PARTITION BY a ORDER BY b) FROM percent_rank_t1;
 a | b |   percent_rank
---+---+------------------
 1 | 1 |                0
 1 | 1 |                0
 1 | 2 | .666666666666667
 1 | 3 |                1
 2 | 4 |                0
 2 | 5 |                1
 3 | 6 |                0
(7 rows)

gaussdb=# DROP TABLE percent_rank_t1;
```

- CUME_DIST()

  Description: The CUME_DIST function is used for generating accumulative distribution sequence numbers for the values in each group. That is, the function calculates the value according to the following formula: Sequence number = Number of rows preceding or peer with current row/Total rows.

  Return type: double precision

  Example:

```
gaussdb=# CREATE TABLE cume_dist_t1(a int, b int);

gaussdb=# INSERT INTO cume_dist_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,CUME_DIST() OVER(PARTITION BY a ORDER BY b) FROM cume_dist_t1;
 a | b | cume_dist
---+---+-----------
 1 | 1 |        .5
 1 | 1 |        .5
 1 | 2 |       .75
 1 | 3 |         1
 2 | 4 |        .5
 2 | 5 |         1
 3 | 6 |         1
(7 rows)

gaussdb=# DROP TABLE cume_dist_t1;
```

- NTILE(num_buckets integer)

  Description: The NTILE function is used for equally allocating sequential data sets to the buckets whose quantity is specified by **num_buckets** according to **num_buckets integer** and allocating the bucket number to each row. Divide the partition as evenly as possible.

  Return type: integer

  Example:

```
gaussdb=# CREATE TABLE ntile_t1(a int, b int);

gaussdb=# INSERT INTO ntile_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,NTILE(2) OVER(PARTITION BY a ORDER BY b) FROM ntile_t1;
 a | b | ntile
---+---+-------
 1 | 1 |     1
 1 | 1 |     1
 1 | 2 |     2
```



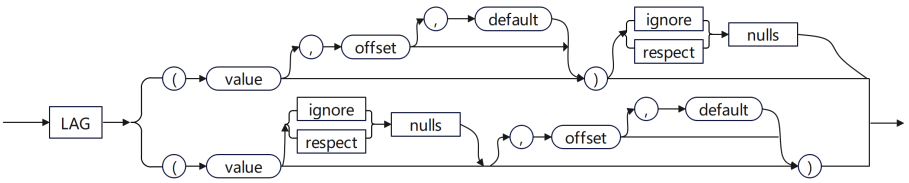
```
 1 | 3 |    2
 2 | 4 |    1
 2 | 5 |    2
 3 | 6 |    1
(7 rows)

gaussdb=# DROP TABLE ntile_t1;
```

- LAG

  Description: The LAG function is used for generating lag values for the corresponding values in each group. That is, the value of the row obtained by moving forward the row corresponding to the current value by **offset** (integer) is the sequence number. If the row does not exist after the moving, the result value is the default value. If not specified, **offset** defaults to **1** and **default** to **NULL**. The type of the **default** value must be the same as that of the **value** value.

  Syntax:

```
LAG(value any [, offset integer [, default any ]])
LAG(value any ignore|respect nulls [, offset integer [, default any ]])
LAG(value any [, offset integer [, default any ]]) ignore|respect nulls
```



  ignore|respect nulls determines whether **NULL** is included in the forward offset value. If not specified, the default value **respect nulls** is used. If **ignore nulls** is specified and **value** is set to **NULL**, **NULL** is not included in the forward offset value. If **ignore nulls** is specified, the function performance deteriorates.

  Return type: same as the parameter type

  Example 1: Disable **ignore nulls** and set **offset** to **3** and **default** to **NULL**.

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE ta1 (hire_date date, last_name varchar(20), department_id int);
CREATE TABLE
gaussdb=# INSERT INTO ta1 VALUES('07-DEC-02', 'Raphaely', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 VALUES('24-JUL-05', 'Tobias',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 VALUES('24-DEC-05', 'Baida',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 VALUES('18-MAY-03', 'Khoo', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('15-NOV-06', 'Himuro',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-AUG-07', 'Colmenares',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-MAY-07', 'yq',  11);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-MAY-08', 'zi',  11);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('', 'yq1',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values(null, 'yq2',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-DEC-07', 'yq3',  30);
```



```
INSERT 0 1
gaussdb=# INSERT INTO ta1 values(null, null,  11);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values(null, null,  11);
INSERT 0 1

-- Call LAG and set offset to 3 and default to null.
gaussdb=# SELECT hire_date, last_name, department_id, lag(hire_date, 3, null)  OVER (PARTITION BY
department_id ORDER BY last_name) AS "NextHired" FROM ta1 ORDER BY department_id;
     hire_date      | last_name  | department_id |      NextHired
--------------------+------------+---------------+---------------------
 2007-05-10 00:00:00 | yq         |            11 |
 2008-05-10 00:00:00 | zi         |            11 |
                     |            |            11 |
                     |            |            11 | 2007-05-10 00:00:00
 2005-12-24 00:00:00 | Baida      |            30 |
 2007-08-10 00:00:00 | Colmenares |            30 |
 2006-11-15 00:00:00 | Himuro     |            30 |
 2003-05-18 00:00:00 | Khoo       |            30 | 2005-12-24 00:00:00
 2002-12-07 00:00:00 | Raphaely   |            30 | 2007-08-10 00:00:00
 2005-07-24 00:00:00 | Tobias     |            30 | 2006-11-15 00:00:00
                     | yq1        |            30 | 2003-05-18 00:00:00
                     | yq2        |            30 | 2002-12-07 00:00:00
 2007-12-10 00:00:00 | yq3        |            30 | 2005-07-24 00:00:00
(13 rows)
```

Example 2: Enable **ignore nulls** and set **offset** to **3** and **default** to **'01-JAN-00'**.

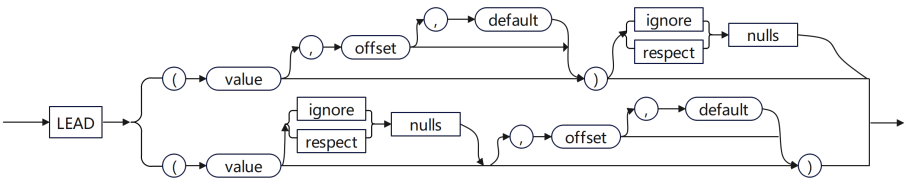
```
gaussdb=# SELECT hire_date, last_name, department_id, lag(hire_date, 3, '01-JAN-00') ignore nulls
OVER (PARTITION BY department_id ORDER BY last_name) AS "NextHired"
 FROM ta1 ORDER BY department_id;
     hire_date      | last_name  | department_id |      NextHired
--------------------+------------+---------------+---------------------
 2007-05-10 00:00:00 | yq         |            11 | 2000-01-01 00:00:00
 2008-05-10 00:00:00 | zi         |            11 | 2000-01-01 00:00:00
                     |            |            11 | 2000-01-01 00:00:00
                     |            |            11 | 2000-01-01 00:00:00
 2005-12-24 00:00:00 | Baida      |            30 | 2000-01-01 00:00:00
 2007-08-10 00:00:00 | Colmenares |            30 | 2000-01-01 00:00:00
 2006-11-15 00:00:00 | Himuro     |            30 | 2000-01-01 00:00:00
 2003-05-18 00:00:00 | Khoo       |            30 | 2005-12-24 00:00:00
 2002-12-07 00:00:00 | Raphaely   |            30 | 2007-08-10 00:00:00
 2005-07-24 00:00:00 | Tobias     |            30 | 2006-11-15 00:00:00
                     | yq1        |            30 | 2003-05-18 00:00:00
                     | yq2        |            30 | 2003-05-18 00:00:00
 2007-12-10 00:00:00 | yq3        |            30 | 2003-05-18 00:00:00
(13 rows)

-- Delete the table.
gaussdb=# DROP TABLE ta1;
DROP TABLE
```

- LEAD

  Description: The LEAD function is used for generating leading values for the corresponding values in each group. That is, the value of the row obtained by moving backward the row corresponding to the current value by **offset** (integer) is the sequence number. If the row after the moving exceeds the total number of rows for the current group, the result value is the default value. If omitted, **offset** defaults to **1** and **default** to **NULL**. The type of the **default** value must be the same as that of the **value** value.

  Syntax:

  ```
  LEAD(value any [, offset integer [, default any ]])
  LEAD(value any ignore|respect nulls [, offset integer [, default any ]])
  LEAD(value any [, offset integer [, default any ]]) ignore|respect nulls
  ```

ignore|respect nulls determines whether **NULL** is included in the backward offset value. If not specified, the default value **respect nulls** is used. If **ignore nulls** is specified and **value** is set to **NULL**, **NULL** is not included in the backward offset value. If **ignore nulls** is specified, the function performance deteriorates.

Return type: same as the parameter type

Example 1: Disable **ignore nulls**, set **offset** to **2**, and do not specify **default**.

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE ta1 (hire_date date, last_name varchar(20), department_id int);
CREATE TABLE
gaussdb=# INSERT INTO ta1 values('07-DEC-02', 'Raphaely', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('24-JUL-05', 'Tobias',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('24-DEC-05', 'Baida',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('18-MAY-03', 'Khoo', 30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('15-NOV-06', 'Himuro',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-AUG-07', 'Colmenares',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-MAY-07', 'yq',  11);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-MAY-08', 'zi',  11);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('', 'yq1',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values(null, 'yq2',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values('10-DEC-07', 'yq3',  30);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values(null, null,  11);
INSERT 0 1
gaussdb=# INSERT INTO ta1 values(null, null,  11);
INSERT 0 1

-- Call LEAD and set offset to 2.
gaussdb=# SELECT hire_date, last_name, department_id, lead(hire_date, 2) OVER (PARTITION BY
department_id ORDER BY last_name) AS "NextHired" FROM ta1 ORDER BY department_id;
     hire_date      | last_name  | department_id |      NextHired
--------------------+------------+---------------+---------------------
 2007-05-10 00:00:00 | yq         |            11 |
 2008-05-10 00:00:00 | zi         |            11 |
                     |            |            11 |
                     |            |            11 |
 2005-12-24 00:00:00 | Baida      |            30 | 2006-11-15 00:00:00
 2007-08-10 00:00:00 | Colmenares |            30 | 2003-05-18 00:00:00
 2006-11-15 00:00:00 | Himuro     |            30 | 2002-12-07 00:00:00
 2003-05-18 00:00:00 | Khoo       |            30 | 2005-07-24 00:00:00
 2002-12-07 00:00:00 | Raphaely   |            30 |
 2005-07-24 00:00:00 | Tobias     |            30 |
                     | yq1        |            30 | 2007-12-10 00:00:00
                     | yq2        |            30 |
```



```
 2007-12-10 00:00:00 | yq3        |            30 |
(13 rows)
```

Example 2: Enable **ignore nulls** and set **offset** to **2** and **default** to **'01-JAN-00'**.

```
gaussdb=# SELECT hire_date, last_name, department_id, lead(hire_date, 2, '01-JAN-00') ignore nulls
OVER (PARTITION BY department_id ORDER BY last_name) AS "NextHired" FROM ta1 ORDER BY
department_id;
     hire_date      | last_name  | department_id |      NextHired
--------------------+------------+---------------+---------------------
 2007-05-10 00:00:00 | yq         |            11 | 2000-01-01 00:00:00
 2008-05-10 00:00:00 | zi         |            11 | 2000-01-01 00:00:00
                     |            |            11 | 2000-01-01 00:00:00
                     |            |            11 | 2000-01-01 00:00:00
 2005-12-24 00:00:00 | Baida      |            30 | 2006-11-15 00:00:00
 2007-08-10 00:00:00 | Colmenares |            30 | 2003-05-18 00:00:00
 2006-11-15 00:00:00 | Himuro     |            30 | 2002-12-07 00:00:00
 2003-05-18 00:00:00 | Khoo       |            30 | 2005-07-24 00:00:00
 2002-12-07 00:00:00 | Raphaely   |            30 | 2007-12-10 00:00:00
 2005-07-24 00:00:00 | Tobias     |            30 | 2000-01-01 00:00:00
                     | yq1        |            30 | 2000-01-01 00:00:00
                     | yq2        |            30 | 2000-01-01 00:00:00
 2007-12-10 00:00:00 | yq3        |            30 | 2000-01-01 00:00:00
(13 rows)

-- Delete the table.
gaussdb=# DROP TABLE ta1;
DROP TABLE
```

- FIRST_VALUE(value any)

  Description: Returns the first value of each group.

  Return type: same as the parameter type

  Example:

```
gaussdb=# CREATE TABLE first_value_t1(a int, b int);

gaussdb=# INSERT INTO first_value_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,FIRST_VALUE(b) OVER(PARTITION BY a ORDER BY b) FROM first_value_t1;
 a | b | first_value
---+---+-------------
 1 | 1 |           1
 1 | 1 |           1
 1 | 2 |           1
 1 | 3 |           1
 2 | 4 |           4
 2 | 5 |           4
 3 | 6 |           6
(7 rows)

gaussdb=# DROP TABLE first_value_t1;
```

- LAST_VALUE(value any)

  Description: Returns the last value of each group.

  Return type: same as the parameter type

  Example:

```
gaussdb=# CREATE TABLE last_value_t1(a int, b int);

gaussdb=# INSERT INTO last_value_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,LAST_VALUE(b) OVER(PARTITION BY a ORDER BY b) FROM last_value_t1;
 a | b | last_value
---+---+------------
 1 | 1 |          1
 1 | 1 |          1
 1 | 2 |          2
```



```
 1 | 3 |         3
 2 | 4 |         4
 2 | 5 |         5
 3 | 6 |         6
(7 rows)

gaussdb=# DROP TABLE last_value_t1;
```

- DELTA

  Description: Returns the difference between the current row and the previous row.

  Parameter: numeric

  Return type: numeric

- RATIO_TO_REPORT(column_name)

  Description: Calculates the ratio of the value in a column to the total value in the group to which the column belongs.

  Parameter: numeric type, or any type that can be implicitly converted to the numeric type.

  Return type: For input parameter types float4 and float8, the return type is of the same value type. For other input parameter types, the return type is numeric.

  NOTE

  When RATIO_TO_REPORT(column_name) is used together with OVER(), the input parameters of OVER() support only **PARTITION BY** and **NULL**.

  Example 1:

```
gaussdb=# CREATE TABLE ratio_to_report_t1(a int, b int);

gaussdb=# INSERT INTO ratio_to_report_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,RATIO_TO_REPORT(b) OVER(PARTITION BY a) FROM ratio_to_report_t1;
 a | b |    ratio_to_report
---+---+------------------------
 1 | 1 |  .1428571428571428571
 1 | 1 |  .1428571428571428571
 1 | 2 |  .2857142857142857143
 1 | 3 |  .4285714285714285714
 2 | 4 |  .4444444444444444444
 2 | 5 |  .5555555555555555556
 3 | 6 | 1.0000000000000000000
(7 rows)

gaussdb=# DROP TABLE ratio_to_report_t1;
```

  Example 2: Nest this function with other functions.

```
gaussdb=# CREATE TABLE ratio_to_report_t1(a int, b int);

gaussdb=# INSERT INTO ratio_to_report_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,TO_CHAR(RATIO_TO_REPORT(b) OVER(PARTITION BY a), '$999eeee') FROM
ratio_to_report_t1;
 a | b | to_char
---+---+---------
 1 | 1 |  1e-01
 1 | 1 |  1e-01
 1 | 2 |  3e-01
 1 | 3 |  4e-01
 2 | 4 |  4e-01
 2 | 5 |  6e-01
 3 | 6 |  1e+00
(7 rows)
```



```
gaussdb=# DROP TABLE ratio_to_report_t1;
```

Example 3: Invoke a stored procedure.

```
gaussdb=# CREATE TABLE ratio_to_report_t1(a int, b int);

gaussdb=# INSERT INTO ratio_to_report_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# CREATE OR REPLACE PROCEDURE ratio_to_report_proc IS CURSOR cur_1 IS SELECT
a,b,RATIO_TO_REPORT(b) OVER(PARTITION BY a) FROM ratio_to_report_t1;
BEGIN
FOR cur IN cur_1 LOOP
RAISE INFO '%', cur.ratio_to_report;
END LOOP;
END;
/

gaussdb=# CALL RATIO_TO_REPORT_PROC();
INFO:  .14285714285714285714
INFO:  .14285714285714285714
INFO:  .28571428571428571429
INFO:  .42857142857142857143
INFO:  .44444444444444444444
INFO:  .55555555555555555556
INFO:  1.00000000000000000000
 ratio_to_report_proc
----------------------

(1 row)

gaussdb=# DROP PROCEDURE ratio_to_report_proc;

gaussdb=# DROP TABLE ratio_to_report_t1;
```

- NTH_VALUE(value any, nth integer)

  Description: The *n*th row for a group is the returned value. If the row does not exist, **NULL** is returned by default.

  Return type: same as the parameter type

  Example:

```
gaussdb=# CREATE TABLE nth_value_t1(a int, b int);

gaussdb=# INSERT INTO nth_value_t1 VALUES(1,1),(1,1),(1, 2),(1, 3),(2, 4),(2, 5),(3,6);

gaussdb=# SELECT a,b,NTH_VALUE(b, 2) OVER(PARTITION BY a order by b) FROM nth_value_t1;
 a | b | nth_value
---+---+-----------
 1 | 1 |         1
 1 | 1 |         1
 1 | 2 |         1
 1 | 3 |         1
 2 | 4 |
 2 | 5 |         5
 3 | 6 |
(7 rows)

gaussdb=# DROP TABLE nth_value_t1;
```

7.6.20 Security Functions

Security Functions

- gs_encrypt_aes128(encryptstr,keystr)

  Description: Encrypts **encryptstr** strings using **keystr** as the encryption password and returns encrypted strings. The value of **keystr** ranges from 8 to

16 bytes and contains at least three types of the following characters: uppercase letters, lowercase letters, digits, and special characters.

Return type: text

Length of the return value: At least 92 bytes and no more than (4*[*Len*/3]+68) bytes, where *Len* indicates the length of the data before encryption (unit: byte).

Example:

```
gaussdb=# SELECT gs_encrypt_aes128('MPPDB','Asdf1234');

                            gs_encrypt_aes128
--------------------------------------------------------------------------------
kbJdlK5LSlAzqNjVvuHUPCphlkuSeD5n2Hel0HiGaSbDQdCmsz1Ky5ZneOurUr56/jtE/U
+BmCw9BgTR2wjQD9440m8=
(1 row)
```

NOTE

A decryption password is required during the execution of this function. For security purposes, the gsql tool does not record the SQL statements containing the function name in the execution history. That is, the execution history of this function cannot be found in **gsql** by paging up and down.

- gs_encrypt(encryptstr,keystr, encrypttype)

  Description: Encrypts **encryptstr** strings using **keystr** as the encryption password and returns encrypted strings based on **encrypttype**.

  Return type: text

Parameter	Type	Description	Value Range
encryptstr	text	Data to be encrypted	-
keystr	text	Encryption password	The value ranges from 8 to 16 bytes and contains at least three types of the following characters: uppercase letters, lowercase letters, digits, and special characters.
encrypttype	text	Encryption/Decryption type (case-insensitive)	aes128, sm4, aes128_cbc_sha256, aes256_cbc_sha256, aes128_gcm_sha256, aes256_gcm_sha256, and sm4_ctr_sm3

Example:

```
gaussdb=#  SELECT gs_encrypt('MPPDB', 'Asdf1234', 'sm4');
       gs_encrypt
 ----------------------------
 ZBzOmaGA4Bb+coyucJ0B8AkIShqc
(1 row)
```

NOTE

A decryption password is required during the execution of this function. For security purposes, the gsql tool does not record the SQL statements containing the function name in the execution history. That is, the execution history of this function cannot be found in **gsql** by paging up and down.

aes128 and **sm4** are compatible with earlier versions. The **aes128** encryption and decryption algorithm in CBC mode is used for AES128 encryption and decryption, and **SHA1** is used for integrity check. The **sm4** encryption and decryption algorithm uses the SM4 CTR mode and does not perform integrity check.

– gs_encrypt_bytea(encryptstr, keystr, encrypttype)

Description: Encrypts **encryptstr** strings using **keystr** as the encryption password and returns encrypted strings based on **encrypttype**.

Return type: bytea

Parameter	Type	Description	Value Range
encryptstr	text	Data to be encrypted	-
keystr	text	Encryption password	The value ranges from 8 to 16 bytes and contains at least three types of the following characters: uppercase letters, lowercase letters, digits, and special characters.
encrypttype	text	Encryption/ Decryption type (case-insensitive)	aes128_cbc_sha256, aes256_cbc_sha256, aes128_gcm_sha256, aes256_gcm_sha256, and sm4_ctr_sm3

Example:

```
gaussdb=#  SELECT gs_encrypt_bytea('MPPDB', 'Asdf1234', 'sm4_ctr_sm3');
                                    gs_encrypt_bytea

------------------------------------------------------------------------------------------------------------------------
--

\x90e286971c2c70410def0a2814af4ac44c737926458b66271d9d1547bc937395ca018d7755672fa
9dc3cdc6ec4a76001dc0e137f3bc5c8a5c51143561f1d09a848bfdebfec5e
(1 row)
```

NOTE

A decryption password is required during the execution of this function. For security purposes, the gsql tool does not record the SQL statements containing the function name in the execution history. That is, the execution history of this function cannot be found in **gsql** by paging up and down.

- gs_decrypt_aes128(decryptstr,keystr)

  Description: Decrypts **decrypt** strings using **keystr** as the decryption password and returns decrypted strings. The **keystr** used for decryption must be consistent with that used for encryption. **keystr** cannot be empty.

> **NOTE**
>
> This parameter needs to be used with the **gs_encrypt_aes128** encryption function.

Return type: text

Example:

```
gaussdb=# SELECT
gs_decrypt_aes128('kbJdlK5LSlAzqNjVvuHUPCphlkuSeD5n2Hel0HiGaSbDQdCmsz1Ky5ZneOurUr56/jtE/U+BmCw9BgTR2wjQD9440m8=','Asdf1234');
 gs_decrypt_aes128
-------------------
 MPPDB
(1 row)
```

> **NOTE**
>
> A decryption password is required during the execution of this function. For security purposes, the gsql tool does not record the SQL statements containing the function name in the execution history. That is, the execution history of this function cannot be found in gsql by paging up and down.

- gs_decrypt(decryptstr, keystr, decrypttype)

  Description: Decrypts **decrypt** strings using **keystr** as the decryption password and returns decrypted strings based on **decrypttype**. **decrypttype** and **keystr** used for decryption must be consistent with **encrypttype** and **keystr** used for encryption. **keystr** cannot be empty.

  This function needs to be used with the **gs_encrypt** encryption function.

  Return type: text

Parameter	Type	Description	Value Range
decryptstr	text	Data to be encrypted	-
keystr	text	Decryption password	The value ranges from 8 to 16 bytes and contains at least three types of the following characters: uppercase letters, lowercase letters, digits, and special characters.
decrypttype	text	Encryption/ Decryption type (case-insensitive)	aes128, sm4, aes128_cbc_sha256, aes256_cbc_sha256, aes128_gcm_sha256, aes256_gcm_sha256, and sm4_ctr_sm3

  Example:

```
gaussdb=# SELECT gs_decrypt('ZBzOmaGA4Bb+coyucJ0B8AkIShqc', 'Asdf1234', 'sm4');
 gs_decrypt
------------
 MPPDB
(1 row)
```


NOTE

A decryption password is required during the execution of this function. For security purposes, the gsql tool does not record the SQL statements containing the function name in the execution history. That is, the execution history of this function cannot be found in gsql by paging up and down.

aes128 and **sm4** are compatible with earlier versions. The **aes128** encryption and decryption algorithm in CBC mode is used for AES128 encryption and decryption, and **SHA1** is used for integrity check. The **sm4** encryption and decryption algorithm uses the SM4 CTR mode and does not perform integrity check.

– gs_decrypt_bytea(decryptstr, keystr, decrypttype)

Description: Decrypts **decrypt** strings using **keystr** as the decryption password and returns decrypted strings based on **decrypttype**. **decrypttype** and **keystr** used for decryption must be consistent with **encrypttype** and **keystr** used for encryption. **keystr** cannot be empty.

This function needs to be used with the **gs_encrypt_bytea** encryption function.

Return type: text

Parameter	Type	Description	Value Range
decryptstr	bytea	Data to be encrypted	-
keystr	text	Decryption password	The value ranges from 8 to 16 bytes and contains at least three types of the following characters: uppercase letters, lowercase letters, digits, and special characters.
decrypttype	text	Encryption/ Decryption type (case-insensitive)	aes128_cbc_sha256, aes256_cbc_sha256, aes128_gcm_sha256, aes256_gcm_sha256, and sm4_ctr_sm3

Example:

```
gaussdb=# SELECT
gs_decrypt_bytea('\x90e286971c2c70410def0a2814af4ac44c737926458b66271d9d1547bc937395
ca018d7755672fa9dc3cdc6ec4a76001dc0e137f3bc5c8a5c51143561f1d09a848bfdebfec5e',
'Asdf1234', 'sm4_ctr_sm3');
 gs_decrypt_bytea
------------------
 MPPDB
(1 row)
```

NOTE

A decryption password is required during the execution of this function. For security purposes, the gsql tool does not record the SQL statements containing the function name in the execution history. That is, the execution history of this function cannot be found in gsql by paging up and down.

- aes_encrypt(str, key_str, init_vector)

Description: Encrypts the string **str** using the encryption password **key_str** and initialization vector **init_vector** based on the AES algorithm.

Parameters in the command above are as follows:

- **str**: character string to be encrypted. If **str** is set to **NULL**, the function returns **NULL**.
- **key_str**: encryption password. If **key_str** is set to **NULL**, the function returns **NULL**. For security purposes, you are advised to use a 128-bit, 192-bit, or 256-bit secure random number as the key character string if the key length is 128 bits, 192 bits, or 256 bits (determined by the value of **block_encryption_mode**).
- **init_vector**: An initialization variable is provided for the required block encryption mode. The length is greater than or equal to 16 bytes. Bytes greater than 16 bytes are automatically ignored. If neither **str** nor **key_str** is **NULL**, this parameter cannot be **NULL**. Otherwise, an error is reported. For security purposes, you are advised to ensure that the IV value for each encryption is unique in OFB mode and that the IV value for each encryption is unpredictable in CBC or CFB mode.

Return type: text

Example:

```
gaussdb=# SELECT aes_encrypt('huwei123','123456vfhex4dyu,vdaladhjsadad','1234567890123456');
 aes_encrypt
-------------
 u*8\x05c?0
(1 row)
```

NOTE

- This function is valid only when GaussDB is compatible with the MY type (that is, **sql_compatibility** is set to **'B'**).
- An encryption password is required during the execution of this function. For security purposes, the gsql tool does not record the SQL statements containing the function name in the execution history. That is, the execution history of this function cannot be found in gsql by paging up and down.
- Do not call this function during operations related to stored procedures, preventing the risk of sensitive information disclosure. In addition, when using the stored procedure that contains the function, you are advised to filter the parameter information of the function before providing the information for external maintenance personnel to locate the fault. Delete the logs after using them.
- Do not call the function when **debug_print_plan** is set to **on**, preventing the risk of sensitive information disclosure. You are advised to filter parameter information of the function in the log files generated when **debug_print_plan** is set to **on** before providing the log files to external maintenance engineers for fault locating. After you finish using the logs, delete them as soon as possible.
- The SQL_ASCII setting performs quite differently from other settings. If the character set of the server is SQL_ASCII, the server interprets the byte values 0 to 127 according to the ASCII standard. The byte values 128 to 255 are regarded as the characters that cannot be parsed. If this parameter is set to SQL_ASCII, no code conversion occurs. When this function calls the third-party OpenSSL library, the returned data is non-ASCII data. Therefore, when the character set of the database server is set to SQL_ASCII, the encoding of the client must also be set to SQL_ASCII. Otherwise, an error is reported. The database does not convert or verify non-ASCII characters.

- aes_decrypt(pass_str, key_str, init_vector)

  Description: Decrypts the string **str** using the decryption password **key_str** and initialization vector **init_vector** based on the AES algorithm.

Parameters in the command above are as follows:

- **pass_str**: character string to be decrypted. If **pass_str** is set to **NULL**, the function returns **NULL**.
- **key_str**: decryption password. If **key_str** is set to **NULL**, the function returns **NULL**. For security purposes, you are advised to use a 128-bit, 192-bit, or 256-bit secure random number as the key character string if the key length is 128 bits, 192 bits, or 256 bits (determined by the value of **block_encryption_mode**).
- **init_vector**: An initialization variable is provided for the required block decryption mode. The length is greater than or equal to 16 bytes. Bytes greater than 16 bytes are automatically ignored. If neither **pass_str** nor **key_str** is **NULL**, this parameter cannot be **NULL**. Otherwise, an error is reported. For security purposes, you are advised to ensure that the IV value for each encryption is unique in OFB mode and that the IV value for each encryption is unpredictable in CBC or CFB mode.

Return type: text

Example:

```
gaussdb=# SELECT
aes_decrypt(aes_encrypt('huwei123','123456vfhex4dyu,vdaladhjsadad','1234567890123456'),'123456vf
hex4dyu,vdaladhjsadad','1234567890123456');
 aes_decrypt
-------------
 huwei123
(1 row)
```


NOTE

- This function is valid only when GaussDB is compatible with the MY type (that is, **sql_compatibility** is set to **'B'**).
- A decryption password is required during the execution of this function. For security purposes, the gsql tool does not record the SQL statements containing the function name in the execution history. That is, the execution history of this function cannot be found in gsql by paging up and down.
- Do not call this function during operations related to stored procedures, preventing the risk of sensitive information disclosure. In addition, when using the stored procedure that contains the function, you are advised to filter the parameter information of the function before providing the information for external maintenance personnel to locate the fault. Delete the logs after using them.
- Do not call the function when **debug_print_plan** is set to **on**, preventing the risk of sensitive information disclosure. You are advised to filter parameter information of the function in the log files generated when **debug_print_plan** is set to **on** before providing the log files to external maintenance engineers for fault locating. After you finish using the logs, delete them as soon as possible.
- To ensure successful decryption, ensure that the values of **block_encryption_mode**, **key_str** and IV are the same as those during encryption.
- Due to encoding differences, encrypted data cannot be directly copied from the gsql client for decryption. In this scenario, the decryption result may not be the character string before encryption.
- The SQL_ASCII setting performs quite differently from other settings. If the character set of the server is SQL_ASCII, the server interprets the byte values 0 to 127 according to the ASCII standard. The byte values 128 to 255 are regarded as the characters that cannot be parsed. If this parameter is set to SQL_ASCII, no code conversion occurs. When this function calls the third-party OpenSSL library, the returned data is non-ASCII data. Therefore, when the character set of the database server is set to SQL_ASCII, the encoding of the client must also be set to SQL_ASCII. Otherwise, an error is reported. The database does not convert or verify non-ASCII characters.

- gs_digest(input_string, hash_algorithm)

  Description: Hashes the input string using the specified hash algorithm and returns a hexadecimal number.

  Parameters in the command above are as follows:

  – **input_string**: character string to be hashed. The value cannot be **NULL**.

  – **hash_algorithm**: specifies the hash algorithm. Currently, SHA-256, SHA-384, SHA-512, and SM3 are supported. Both uppercase and lowercase letters are supported. If an unsupported hash algorithm is used, an error is reported.

  Return type: text

  Example:

```
gaussdb=# SELECT pg_catalog.gs_digest('gaussdb', 'sha256');
                                gs_digest
------------------------------------------------------------------
 4dc50d746f4e04f9b446986b34a0050e358fbfb8bc1fba314c54b52a417b0b8e
(1 row)
```

- gs_password_deadline

  Description: Indicates the number of remaining days before the password of the current user expires.

  Return type: interval

  Example:


```
gaussdb=# SELECT gs_password_deadline();
 gs_password_deadline
----------------------
 83 days 17:44:32.196094
(1 row)
```

- gs_password_notifytime()

  Description: Specifies the number of days prior to password expiration that a user will receive a reminder.

  Return type: int32

- login_audit_messages(BOOLEAN)

  Description: Queries login information about a login user.

  Return type: tuple

  Example:

  – Check the date, time, and IP address of the last successful login.

  ```
  gaussdb=> SELECT * FROM login_audit_messages(true);
   username | database |       logintime        |    mytype     | result | client_conninfo
  ----------+----------+------------------------+---------------+--------+-----------------
   omm      | testdb | 2020-06-29 21:56:40+08 | login_success | ok     | gsql@[local]
  (1 row)
  ```

  – Check the number, date, and time of failed attempts since the previous successful login.

  ```
  gaussdb=>  SELECT * FROM login_audit_messages(false);
   username | database |       logintime        |    mytype    | result |  client_conninfo
  ----------+----------+------------------------+--------------+--------+-------------------
   omm      | testdb | 2020-06-29 21:57:55+08 | login_failed | failed | [unknown]@[local]
   omm      | testdb | 2020-06-29 21:57:53+08 | login_failed | failed | [unknown]@[local]
  (2 rows)
  ```

- login_audit_messages_pid

  Description: Queries login information about a login user. Different from **login_audit_messages**, this function queries login information based on **backendid**. Information about subsequent logins of the same user does not alter the query result of previous logins and cannot be found using this function.

  Return type: tuple

  NOTE

  When the thread pool is enabled, the **backendid** obtained in the same session may change due to thread switchover. As a result, the return values are different when the function is called for multiple times. You are advised not to call this function when the thread pool is enabled.

  Example:

  – Check the date, time, and IP address of the last successful login.

  ```
  gaussdb=> SELECT * FROM login_audit_messages_pid(true);
   username | database |       logintime        |    mytype     | result | client_conninfo |    backendid
  ----------+----------+------------------------+---------------+--------+-----------------+-----------------
   omm      | testdb | 2020-06-29 21:56:40+08 | login_success | ok     | gsql@[local]    |
  139823109633792
  (1 row)
  ```

  – Check the number, date, and time of failed attempts since the previous successful login.

  ```
  gaussdb=> SELECT * FROM login_audit_messages_pid(false);
   username | database |       logintime        |    mytype    | result |  client_conninfo  |    backendid
  ----------+----------+------------------------+--------------+--------+-------------------+-----------------
   omm      | testdb | 2020-06-29 21:57:55+08 | login_failed | failed | [unknown]@[local] |
  139823109633792
  ```

```
 omm      | testdb | 2020-06-29 21:57:53+08 | login_failed | failed | [unknown]@[local] |
139823109633792
(2 rows)
```

- inet_server_addr

  Description: Displays the server IP address.

  Return type: inet

  Example:

```
gaussdb=# SELECT inet_server_addr();
 inet_server_addr
------------------
 10.10.0.13
(1 row)
```

  NOTE

  - The client IP address 10.10.0.50 and server IP address 10.10.0.13 are used as an example.
  - If the database is connected to the local PC, the value is empty.

- inet_client_addr

  Description: Displays the client IP address.

  Return type: inet

  Example:

```
gaussdb=# SELECT inet_client_addr();
 inet_client_addr
------------------
 10.10.0.50
(1 row)
```

  NOTE

  - The client IP address 10.10.0.50 and server IP address 10.10.0.13 are used as an example.
  - If the database is connected to the local PC, the value is empty.

- pg_query_audit

  Description: Views audit logs of the primary database node.

  Return type: record

  The following table describes return columns.

Name	Type	Description
time	timestamp with time zone	Operation time
type	text	Operation
result	text	Operation result
userid	oid	User ID
username	text	Name of the user who performs the operation
database	text	Database name

Name	Type	Description
client_conninfo	text	Client connection information
object_name	text	Object name
detail_info	text	Operation details
node_name	text	Node name
thread_id	text	Thread ID
local_port	text	Local port
remote_port	text	Remote port

- pg_delete_audit()

  Description: Deletes audit logs in a specified period.

  Return type: void

- alldigitsmasking

  Description: Specifies the internal function of the masking policy, which is used to anonymize all characters.

  Parameter: col text, letter character default '0'

  Return type: text

- creditcardmasking

  Description: Specifies the internal function of the masking policy, which is used to anonymize all credit card information.

  Parameter: col text, letter character default 'x'

  Return type: text

- randommasking

  Description: Specifies the internal function of the masking policy. The random policy is used.

  Parameter: col text

  Return type: text

- fullemailmasking

  Description: Specifies the internal function of the masking policy, which is used to anonymize the text (except **@**) before the last period (.).

  Parameter: col text, letter character default 'x'

  Return type: text

- basicemailmasking

  Description: Specifies the internal function of the masking policy, which is used to anonymize the text before the first at sign (@).

  Parameter: col text, letter character default 'x'

  Return type: text

- shufflemasking

Description: Specifies the internal function of the masking policy, which is used to sort characters out of order.

Parameter: col text

Return type: text

7.6.21 Ledger Database Functions

- get_dn_hist_relhash(text, text)

  Description: Returns the hash value of table-level data in a specified tamper-proof user table. This function is not supported in the centralized mode.

  Parameter type: text

  Return type: hash16

- ledger_hist_check(text, text)

  Description: Verifies the consistency between the hash value of table-level data in a specified tamper-proof user table and that in the corresponding history table.

  Parameter type: text

  Return type: Boolean

  Example:

  ```
  -- Create a schema.
  gaussdb=# CREATE SCHEMA ledgernsp WITH BLOCKCHAIN;
  -- Create a table.
  gaussdb=# CREATE TABLE ledgernsp.tab(a int, b text);
  -- Insert data.
  gaussdb=# INSERT INTO ledgernsp.tab values(generate_series(1, 10000), 'test');
  gaussdb=# SELECT ledger_hist_check('ledgernsp','tab');
   ledger_hist_check
  -------------------
   t
  (1 row)
  ```

- ledger_hist_repair(text, text)

  Description: Restores the hash value of the history table corresponding to the specified tamper-proof user table to be the same as that of the user table, and returns the hash difference.

  Parameter type: text

  Return type: hash16

  Example:

  ```
  gaussdb=# SELECT ledger_hist_repair('ledgernsp','tab');
   ledger_hist_repair
  --------------------
   0000000000000000
  (1 row)
  ```

- ledger_hist_archive(text, text)

  Description: Archives the history table corresponding to a specified tamper-proof user table to the **hist_back** folder in the audit log directory.

  Parameter type: text

  Return type: Boolean

  Example:

  ```
  gaussdb=# SELECT ledger_hist_archive('ledgernsp','tab');
   ledger_hist_archive
  ```



```
---------------------
 t
(1 row)
```

- ledger_gchain_check(text, text)

  Description: Verifies the consistency between the history table hash corresponding to the specified tamper-proof user table and the **relhash** corresponding to the global history table.

  Parameter type: text

  Return type: Boolean

  Example:

  ```
  gaussdb=# SELECT ledger_gchain_check('ledgernsp','tab');
   ledger_gchain_check
  ---------------------
   t
  (1 row)
  ```

- ledger_gchain_repair(text, text)

  Description: Restores relhash of a specified tamper-proof user table in the global history table so that the hash is the same as that in the history table, and returns the total hash value of the specified table.

  Parameter type: text

  Return type: hash16

  Example:

  ```
  gaussdb=# SELECT ledger_gchain_repair('ledgernsp','tab');
   ledger_gchain_repair
  ----------------------
   da30c1260af5be50
  (1 row)
  ```

- ledger_gchain_archive(void)

  Description: Archives global history tables to the **hist_back** folder in the audit log directory.

  Parameter type: void

  Return type: Boolean

  Example:

  ```
  gaussdb=# SELECT ledger_gchain_archive();
   ledger_gchain_archive
  -----------------------
   t
  (1 row)
  ```

- hash16in(cstring)

  Description: Converts the input hexadecimal string into the internal hash16 format.

  Parameter type: cstring

  Return type: hash16

- hash16out(hash16)

  Description: Converts internal hash16 data to hexadecimal cstring data.

  Parameter type: hash16

  Return type: cstring

- hash32in(cstring)

Description: Converts the input hexadecimal string (32 characters) into the internal type hash32.

Parameter type: cstring

Return type: hash32

- hash32out(hash32)

  Description: Converts internal hash32 data to hexadecimal cstring data.

  Parameter type: hash32

  Return type: cstring

7.6.22 Encrypted Functions and Operators

- byteawithoutorderwithequalcolin(cstring)

  Description: Converts input data to the internal byteawithoutorderwithequalcol format.

  Parameter type: cstring

  Return type: byteawithoutorderwithequalcol

- byteawithoutorderwithequalcolout(byteawithoutorderwithequalcol)

  Description: Converts internal data of the byteawithoutorderwithequalcol type to data of the cstring type.

  Parameter type: byteawithoutorderwithequalcol

  Return type: cstring

- byteawithoutorderwithequalcolsend(byteawithoutorderwithequalcol)

  Description: Converts data of the byteawithoutorderwithequalcol type to data of the bytea type.

  Parameter type: byteawithoutorderwithequalcol

  Return type: bytea

- byteawithoutorderwithequalcolrecv(internal)

  Description: Converts data of the internal type to data of the byteawithoutorderwithequalcol type.

  Parameter type: internal

  Return type: byteawithoutorderwithequalcol

- byteawithoutorderwithequalcoltypmodin(cstring[])

  Description: Converts data of the cstring[] type to data of the byteawithoutorderwithequalcol type.

  Parameter type: cstring[]

  Return type: int4

- byteawithoutorderwithequalcoltypmodout(int4)

  Description: Converts data of the int4 type into data of the cstring type.

  Parameter type: int4

  Return type: cstring

- byteawithoutordercolin(cstring)

  Description: Converts input data to the internal byteawithoutordercolin format.

Parameter type: cstring

Return type: byteawithoutordercol

- byteawithoutordercolout(byteawithoutordercol)

  Description: Converts internal data of the byteawithoutordercol type to data of the cstring type.

  Parameter type: byteawithoutordercol

  Return type: cstring

- byteawithoutordercolsend(byteawithoutordercol)

  Description: Converts data of the byteawithoutordercol type to data of the bytea type.

  Parameter type: byteawithoutordercol

  Return type: bytea

- byteawithoutordercolrecv(internal)

  Description: Converts data of the internal type to data of the byteawithoutordercol type.

  Parameter type: internal

  Return type: byteawithoutordercol

- byteawithoutorderwithequalcolcmp(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)

  Description: Compares two byteawithoutorderwithequalcol data sizes. If the first data size is smaller than the second one, **–1** is returned. If the first data size is equal to the second one, **0** is returned. If the first data size is larger than the second one, **1** is returned.

  Parameter type: byteawithoutorderwithequalcol, byteawithoutorderwithequalcol

  Return type: int4

- byteawithoutorderwithequalcolcmpbytear(byteawithoutorderwithequalcol, bytea)

  Description: Compares the byteawithoutorderwithequalcol and bytea data sizes. If the first data size is smaller than the second one, **–1** is returned. If the first data size is equal to the second one, **0** is returned. If the first data size is larger than the second one, **1** is returned.

  Parameter type: byteawithoutorderwithequalcol or bytea

  Return type: int4

- byteawithoutorderwithequalcolcmpbyteal(bytea, byteawithoutorderwithequalcol)

  Description: Compares the bytea and byteawithoutorderwithequalcol data sizes. If the first data size is smaller than the second one, **–1** is returned. If the first data size is equal to the second one, **0** is returned. If the first data size is larger than the second one, **1** is returned.

  Parameter type: bytea, byteawithoutorderwithequalcol

  Return type: int4

- byteawithoutorderwithequalcoleq(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)

Description: Compares two byteawithoutorderwithequalcol data records. If they are the same, **true** is returned. Otherwise, **false** is returned.

Parameter type: byteawithoutorderwithequalcol, byteawithoutorderwithequalcol

Return type: Boolean

- byteawithoutorderwithequalcoleqbyteal(bytea, byteawithoutorderwithequalcol)

  Description: Compares the bytea and byteawithoutorderwithequalcol data records. If they are the same, **true** is returned. Otherwise, **false** is returned.

  Parameter type: bytea, byteawithoutorderwithequalcol

  Return type: Boolean

- byteawithoutorderwithequalcoleqbytear(byteawithoutorderwithequalcol, bytea)

  Description: Compares the byteawithoutorderwithequalcol and bytea data records. If they are the same, **true** is returned. Otherwise, **false** is returned.

  Parameter type: byteawithoutorderwithequalcol, bytea

  Return type: Boolean

- byteawithoutorderwithequalcolne(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)

  Description: Compares two byteawithoutorderwithequalcol data records. If they are different, **true** is returned. Otherwise, **false** is returned.

  Parameter type: byteawithoutorderwithequalcol, byteawithoutorderwithequalcol

  Return type: Boolean

- byteawithoutorderwithequalcolnebyteal(bytea, byteawithoutorderwithequalcol)

  Description: Compares the bytea and byteawithoutorderwithequalcol data records. If they are the same, **true** is returned. Otherwise, **false** is returned.

  Parameter type: bytea, byteawithoutorderwithequalcol

  Return type: Boolean

- byteawithoutorderwithequalcolnebytear(byteawithoutorderwithequalcol, bytea)

  Description: Compares the byteawithoutorderwithequalcol and bytea data records. If they are the same, **true** is returned. Otherwise, **false** is returned.

  Parameter type: byteawithoutorderwithequalcol, bytea

  Return type: Boolean

- hll_hash_byteawithoutorderwithequalcol(byteawithoutorderwithequalcol)

  Description: Returns the hll hash value of byteawithoutorderwithequalcol.

  Parameter type: byteawithoutorderwithequalcol

  Return type: hll_hashval

- tee_lt(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)

  Description: Checks whether a value of the byteawithoutorderwithequalcol type is less than another value of the byteawithoutorderwithequalcol type. If yes, **true** is returned. Otherwise, **false** is returned. This function is used for

parsing the encrypted operator "<" only when the memory decryption emergency channel is enabled. This function cannot be directly used.

Parameter type: byteawithoutorderwithequalcol, byteawithoutorderwithequalcol

Return type: Boolean

- tee_gt(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)

  Description: Checks whether a value of the byteawithoutorderwithequalcol type is greater than another value of the byteawithoutorderwithequalcol type. If yes, **true** is returned. Otherwise, **false** is returned. This function is used for parsing the encrypted operator ">" only when the memory decryption emergency channel is enabled. This function cannot be directly used.

  Parameter type: byteawithoutorderwithequalcol, byteawithoutorderwithequalcol

  Return type: Boolean

- tee_le(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)

  Description: Checks whether a value of the byteawithoutorderwithequalcol type is less than or equal to another value of the byteawithoutorderwithequalcol type. If yes, **true** is returned. Otherwise, **false** is returned. This function is used for parsing the encrypted operator "<=" only when the memory decryption emergency channel is enabled. This function cannot be directly used.

  Parameter type: byteawithoutorderwithequalcol, byteawithoutorderwithequalcol

  Return type: Boolean

- tee_ge(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)

  Description: Checks whether a value of the byteawithoutorderwithequalcol type is greater than or equal to another value of the byteawithoutorderwithequalcol type. If yes, **true** is returned. Otherwise, **false** is returned. This function is used for parsing the encrypted operator ">=" only when the memory decryption emergency channel is enabled. This function cannot be directly used.

  Parameter type: byteawithoutorderwithequalcol, byteawithoutorderwithequalcol

  Return type: Boolean

- tee_like(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)

  Description: Checks whether byteawithoutorderwithequalcol data meets fuzzy match. If yes, **true** is returned. Otherwise, **false** is returned. This function cannot be directly used.

  Parameter type: byteawithoutorderwithequalcol, byteawithoutorderwithequalcol

  Return type: Boolean

- tee_nlike(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)

  Description: Checks whether byteawithoutorderwithequalcol data does not meet fuzzy match. If yes, **true** is returned. Otherwise, **false** is returned. This function cannot be directly used.

  Parameter type: byteawithoutorderwithequalcol, byteawithoutorderwithequalcol

Return type: Boolean

- tee_calculation(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)

  Description: General function for mathematical operations between byteawithoutorderwithequalcol types. This function cannot be directly used.

  Parameter type: byteawithoutorderwithequalcol, byteawithoutorderwithequalcol

  Return type: byteawithoutorderwithequalcol

- tee_sortsupport(internal)

  Description: Function used to support sorting, which is internally invoked during sorting and cannot be invoked by users.

  Parameter type: internal

  Return type: void

- teegtsel(internal, oid, internal, integer)

  Description: Selection rate of ciphertext greater than and greater than or equal to. The default value **0.3333333333333333** is returned. This function cannot be invoked by common users.

  Parameter type: internal, oid, internal, integer

  Return type: double precision

- teeltsel(internal, oid, internal, integer)

  Description: Selection rate of ciphertext less than and less than or equal to. The default value **0.3333333333333333** is returned. This function cannot be invoked by common users.

  Parameter type: internal, oid, internal, integer

  Return type: double precision

- teelikesel(internal, oid, internal, integer)

  Description: Selection rate of ciphertext fuzzy match. The default value **0.005** is returned. This function cannot be invoked by common users.

  Parameter type: internal, oid, internal, integer

  Return type: double precision

- teenlikesel(internal, oid, internal, integer)

  Description: Selection rate of ciphertext non-fuzzy-match. The default value **0.005** is returned. This function cannot be invoked by common users.

  Parameter type: internal, oid, internal, integer

  Return type: double precision

- sum(byteawithoutorderwithequalcol)

  Description: Aggregates the ciphertext sum. It is not supported in the current version.

  Parameter type: byteawithoutorderwithequalcol

  Return type: byteawithoutorderwithequalcol

- avg(byteawithoutorderwithequalcol)

  Description: Aggregates the ciphertext average value. It is not supported in the current version.

  Parameter type: byteawithoutorderwithequalcol

Return type: byteawithoutorderwithequalcol

- min(byteawithoutorderwithequalcol)

  Description: Aggregates the ciphertext minimum value. It is not supported in the current version.

  Parameter type: byteawithoutorderwithequalcol

  Return type: byteawithoutorderwithequalcol

- max(byteawithoutorderwithequalcol)

  Description: Aggregates the ciphertext maximum value. It is not supported in the current version.

  Parameter type: byteawithoutorderwithequalcol

  Return type: byteawithoutorderwithequalcol

- stddev_samp(byteawithoutorderwithequalcol)

  Description: Aggregates the standard deviation of ciphertext samples. It is not supported in the current version.

  Parameter type: byteawithoutorderwithequalcol

  Return type: byteawithoutorderwithequalcol

- stddev_pop(byteawithoutorderwithequalcol)

  Description: Aggregates the total standard deviation of ciphertext. It is not supported in the current version.

  Parameter type: byteawithoutorderwithequalcol

  Return type: byteawithoutorderwithequalcol

- var_samp(byteawithoutorderwithequalcol)

  Description: Aggregates the variance of ciphertext samples. It is not supported in the current version.

  Parameter type: byteawithoutorderwithequalcol

  Return type: byteawithoutorderwithequalcol

- var_pop(byteawithoutorderwithequalcol)

  Description: Aggregates the total variance of ciphertext. It is not supported in the current version.

  Parameter type: byteawithoutorderwithequalcol

  Return type: byteawithoutorderwithequalcol

- tee_trans(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)

  Description: Aggregates the internal processing function for ciphertext. This function cannot be invoked by common users.

  Parameter type: byteawithoutorderwithequalcol, byteawithoutorderwithequalcol

  Return type: byteawithoutorderwithequalcol

- tee_collect(byteawithoutorderwithequalcol, byteawithoutorderwithequalcol)

  Description: Aggregates the internal processing function for ciphertext. This function cannot be invoked by common users.

  Parameter type: byteawithoutorderwithequalcol, byteawithoutorderwithequalcol

  Return type: byteawithoutorderwithequalcol

- tee_final(byteawithoutorderwithequalcol)

  Description: Aggregates the internal processing function for ciphertext. This function cannot be invoked by common users.

  Parameter type: byteawithoutorderwithequalcol

  Return type: byteawithoutorderwithequalcol

- security_tee_process()

  Description: Manages the computational state in a secret trusted domain. It is not supported in the current version.

  Parameter type: int

  Return type: Boolean

- ce_encrypt_deterministic(text, oid)

  Description: Encrypts the plaintext of the text type to the encrypted equivalent structure. The key OID is the value of **column_key_distributed_id** in the gs_column_keys system catalog. If the input parameter is **NULL**, **NULL** is returned.

  Parameter type: text, oid

  Note: This is used only when the memory decryption emergency channel is enabled.

  Return type: byteawithoutorderwithequalcol

  For details, see "Migrate plaintext and ciphertext columns to each other." in "Memory Decryption Emergency Channel > Using gsql to Operate the Memory Decryption Emergency Channel" in *Feature Guide*.

- ce_encrypt_deterministic(int1, oid)

  Description: Encrypts the plaintext of the int1 type to the encrypted equivalent structure. The key OID is the value of **column_key_distributed_id** in the gs_column_keys system catalog. If the input parameter is **NULL**, **NULL** is returned.

  Parameter type: int1, oid

  Note: This is used only when the memory decryption emergency channel is enabled.

  Return type: byteawithoutorderwithequalcol

- ce_encrypt_deterministic(int2, oid)

  Description: Encrypts the plaintext of the int2 type to the encrypted equivalent structure. The key OID is the value of **column_key_distributed_id** in the gs_column_keys system catalog. If the input parameter is **NULL**, **NULL** is returned.

  Parameter type: int2, oid

  Note: This is used only when the memory decryption emergency channel is enabled.

  Return type: byteawithoutorderwithequalcol

- ce_encrypt_deterministic(int4, oid)

  Description: Encrypts the plaintext of the int4 type to the encrypted equivalent structure. The key OID is the value of **column_key_distributed_id** in the gs_column_keys system catalog. If the input parameter is **NULL**, **NULL** is returned.

Parameter type: int4, oid

Note: This is used only when the memory decryption emergency channel is enabled.

Return type: byteawithoutorderwithequalcol

- ce_encrypt_deterministic(int8, oid)

  Description: Encrypts the plaintext of the int8 type to the encrypted equivalent structure. The key OID is the value of **column_key_distributed_id** in the gs_column_keys system catalog. If the input parameter is **NULL**, **NULL** is returned.

  Parameter type: int8, oid

  Note: This is used only when the memory decryption emergency channel is enabled.

  Return type: byteawithoutorderwithequalcol

- ce_encrypt_deterministic(float4, oid)

  Description: Encrypts the plaintext of the float4 type to the encrypted equivalent structure. The key OID is the value of **column_key_distributed_id** in the gs_column_keys system catalog. If the input parameter is **NULL**, **NULL** is returned.

  Parameter type: float4, oid

  Note: This is used only when the memory decryption emergency channel is enabled.

  Return type: byteawithoutorderwithequalcol

- ce_encrypt_deterministic(float8, oid)

  Description: Encrypts the plaintext of the float8 type to the encrypted equivalent structure. The key OID is the value of **column_key_distributed_id** in the gs_column_keys system catalog. If the input parameter is **NULL**, **NULL** is returned.

  Parameter type: float8, oid

  Note: This is used only when the memory decryption emergency channel is enabled.

  Return type: byteawithoutorderwithequalcol

- ce_encrypt_deterministic(numeric, oid)

  Description: Encrypts the plaintext of the numeric type to the encrypted equivalent structure. The key OID is the value of **column_key_distributed_id** in the gs_column_keys system catalog. If the input parameter is **NULL**, **NULL** is returned.

  Parameter type: numeric, oid

  Note: This is used only when the memory decryption emergency channel is enabled.

  Return type: byteawithoutorderwithequalcol

- ce_encrypt_deterministic(clob, oid)

  Description: Encrypts the plaintext of the CLOB type to the encrypted equivalent structure. The key OID is the value of **column_key_distributed_id** in the gs_column_keys system catalog. If the input parameter is **NULL**, **NULL** is returned.

  Parameter type: clob, oid

Note: This is used only when the memory decryption emergency channel is enabled.

Return type: byteawithoutorderwithequalcol

- ce_encrypt_deterministic(varchar, oid)

  Description: Encrypts the plaintext of the varchar type to the encrypted equivalent structure. The key OID is the value of **column_key_distributed_id** in the gs_column_keys system catalog. If the input parameter is **NULL**, **NULL** is returned.

  Parameter type: varchar, oid

  Note: This is used only when the memory decryption emergency channel is enabled.

  Return type: byteawithoutorderwithequalcol

- ce_encrypt_deterministic(nvarchar2, oid)

  Description: Encrypts the plaintext of the nvarchar2 type to the encrypted equivalent structure. The key OID is the value of **column_key_distributed_id** in the gs_column_keys system catalog. If the input parameter is **NULL**, **NULL** is returned.

  Parameter type: nvarchar2, oid

  Note: This is used only when the memory decryption emergency channel is enabled.

  Return type: byteawithoutorderwithequalcol

- ce_encrypt_deterministic(bpchar, oid)

  Description: Encrypts the plaintext of the bpchar type to the encrypted equivalent structure. The key OID is the value of **column_key_distributed_id** in the gs_column_keys system catalog. If the input parameter is **NULL**, **NULL** is returned.

  Parameter type: bpchar, oid

  Note: This is used only when the memory decryption emergency channel is enabled.

  Return type: byteawithoutorderwithequalcol

- ce_encrypt_deterministic(nvarchar2, oid)

  Description: Encrypts the plaintext of the nvarchar2 type to the encrypted equivalent structure. The key OID is the value of **column_key_distributed_id** in the gs_column_keys system catalog. If the input parameter is **NULL**, **NULL** is returned.

  Parameter type: nvarchar2, oid

  Note: This is used only when the memory decryption emergency channel is enabled.

  Return type: byteawithoutorderwithequalcol

- ce_encrypt_deterministic(bytea, oid)

  Description: Encrypts the plaintext of the bytea type to the encrypted equivalent structure. The key OID is the value of **column_key_distributed_id** in the gs_column_keys system catalog. If the input parameter is **NULL**, **NULL** is returned.

  Parameter type: bytea, oid

Note: This is used only when the memory decryption emergency channel is enabled.

Return type: byteawithoutorderwithequalcol

- ce_decrypt_deterministic(byteawithoutorderwithequalcol, oid)

  Description: Decrypts the ciphertext of the encrypted equivalent structure to the plaintext data of the text type. The key OID is the value of **column_key_distributed_id** in the gs_column_keys system catalog. If the input parameter is **NULL**, **NULL** is returned.

  Parameter type: byteawithoutorderwithequalcol, oid

  Note: This is used only when the memory decryption emergency channel is enabled.

  Return type: text

  For details, see "Migrate plaintext and ciphertext columns to each other." in "Memory Decryption Emergency Channel > Using gsql to Operate the Memory Decryption Emergency Channel" in *Feature Guide*.

  NOTE

  The following functions starting with ce_encrypt/ce_decrypt exist in the pg_catalog namespace.

- ce_encrypt_text(text, int4, boolean, int4, internal)

  Description: Encryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: text, int4, boolean, int4, internal

  Return type: byteawithoutorderwithequalcol

- ce_encrypt_varchar(varchar, int4, boolean, int4, internal)

  Description: Encryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: varchar, int4, boolean, int4, internal

  Return type: byteawithoutorderwithequalcol

- ce_encrypt_nvarchar2(nvarchar2, int4, boolean, int4, internal)

  Description: Encryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: nvarchar2, int4, boolean, int4, internal

  Return type: byteawithoutorderwithequalcol

- ce_encrypt_bpchar(bpchar, int4, boolean, int4, internal)

  Description: Encryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: bpchar, int4, boolean, int4, internal

  Return type: byteawithoutorderwithequalcol

- ce_encrypt_int1(int1, int4, boolean, int4, internal)

  Description: Encryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: int1, int4, boolean, int4, internal

  Return type: byteawithoutorderwithequalcol

- ce_encrypt_int2(int2, int4, boolean, int4, internal)

  Description: Encryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: int2, int4, boolean, int4, internal

  Return type: byteawithoutorderwithequalcol
- ce_encrypt_int4(int4, int4, boolean, int4, internal)

  Description: Encryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: int4, int4, boolean, int4, internal

  Return type: byteawithoutorderwithequalcol
- ce_encrypt_int8(int8, int4, boolean, int4, internal)

  Description: Encryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: int8, int4, boolean, int4, internal

  Return type: byteawithoutorderwithequalcol
- ce_encrypt_float4(float4, int4, boolean, int4, internal)

  Description: Encryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: float4, int4, boolean, int4, internal

  Return type: byteawithoutorderwithequalcol
- ce_encrypt_float8(float8, int4, boolean, int4, internal)

  Description: Encryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: float8, int4, boolean, int4, internal

  Return type: byteawithoutorderwithequalcol
- ce_encrypt_numeric(numeric, int4, boolean, int4, internal)

  Description: Encryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: numeric, int4, boolean, int4, internal

  Return type: byteawithoutorderwithequalcol
- ce_encrypt_bytea(bytea, int4, boolean, int4, internal)

  Description: Encryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: bytea, int4, boolean, int4, internal

  Return type: byteawithoutorderwithequalcol
- ce_encrypt_clob(clob, int4, boolean, int4, internal)

  Description: Encryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: clob, int4, boolean, int4, internal

  Return type: byteawithoutorderwithequalcol
- ce_decrypt_text(byteawithoutorderwithequalcol, int4, boolean, int4, internal)

  Description: Decryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

Parameter type: byteawithoutorderwithequalcol, int4, boolean, int4, internal

Return type: text

- ce_decrypt_varchar(byteawithoutorderwithequalcol, int4, boolean, int4, internal)

  Description: Decryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: byteawithoutorderwithequalcol, int4, boolean, int4, internal

  Return type: varchar

- ce_decrypt_nvarchar2(byteawithoutorderwithequalcol, int4, boolean, int4, internal)

  Description: Decryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: byteawithoutorderwithequalcol, int4, boolean, int4, internal

  Return type: nvarchar2

- ce_decrypt_bpchar(byteawithoutorderwithequalcol, int4, boolean, int4, internal)

  Description: Decryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: byteawithoutorderwithequalcol, int4, boolean, int4, internal

  Return type: bpchar

- ce_decrypt_int1(byteawithoutorderwithequalcol, int4, boolean, int4, internal)

  Description: Decryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: byteawithoutorderwithequalcol, int4, boolean, int4, internal

  Return type: int1

- ce_decrypt_int2(byteawithoutorderwithequalcol, int4, boolean, int4, internal)

  Description: Decryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: byteawithoutorderwithequalcol, int4, boolean, int4, internal

  Return type: int2

- ce_decrypt_int4(byteawithoutorderwithequalcol, int4, boolean, int4, internal)

  Description: Decryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: byteawithoutorderwithequalcol, int4, boolean, int4, internal

  Return type: int4

- ce_decrypt_int8(byteawithoutorderwithequalcol, int4, boolean, int4, internal)

  Description: Decryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: byteawithoutorderwithequalcol, int4, boolean, int4, internal

  Return type: int8

- ce_decrypt_float4(byteawithoutorderwithequalcol, int4, boolean, int4, internal)

  Description: Decryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

Parameter type: byteawithoutorderwithequalcol, int4, boolean, int4, internal

Return type: float4

- ce_decrypt_float8(byteawithoutorderwithequalcol, int4, boolean, int4, internal)

  Description: Decryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: byteawithoutorderwithequalcol, int4, boolean, int4, internal

  Return type: float8

- ce_decrypt_numeric(byteawithoutorderwithequalcol, int4, boolean, int4, internal)

  Description: Decryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: byteawithoutorderwithequalcol, int4, boolean, int4, internal

  Return type: numeric

- ce_decrypt_bytea(byteawithoutorderwithequalcol, int4, boolean, int4, internal)

  Description: Decryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: byteawithoutorderwithequalcol, int4, boolean, int4, internal

  Return type: bytea

- ce_decrypt_clob(byteawithoutorderwithequalcol, int4, boolean, int4, internal)

  Description: Decryption function for fully-encrypted type conversion. This function cannot be invoked by common users.

  Parameter type: byteawithoutorderwithequalcol, int4, boolean, int4, internal

  Return type: clob

Examples

Encrypted equality functions such as byteawithoutorderwithequalcolin and byteawithoutorderwithequalcolout are read/write format conversion functions such as in, out, send, and recv specified by the data type byteawithoutorderwithequalcol in the database kernel. For details, see the byteain and byteaout functions of the bytea type. However, the local CEK is verified, and the function can be successfully executed only when the encrypted column contains a cekoid that exists on the local host.

```
-- For example, if the int_type encrypted table exists, int_col2 is the encrypted column.

-- Use a non-encrypted client to connect to the database and query the ciphertext of the encrypted column.
gaussdb=# SELECT int_col2 FROM int_type;
                                                            int_col2
------------------------------------------------------------------------------------------------------------------------------------------------

\x01c35301bf421c8edf38c34704bcc82838742917778ccb402a1b7452ad4a6ac7371acc0ac33100000035fe3424
919854c86194f1aa5bb4e1ca656e8fc6d05324a1419b69f488bdc3c6
(1 row)

-- The ciphertext of the encrypted column is used as the input parameter of
byteawithoutorderwithequalcolin. The format is converted from cstring to byteawithoutorderwithequalcol.
gaussdb=# SELECT
byteawithoutorderwithequalcolin('\x01c35301bf421c8edf38c34704bcc82838742917778ccb402a1b7452ad4a
```

```
6ac7371acc0ac33100000035fe3424919854c86194f1aa5bb4e1ca656e8fc6d05324a1419b69f488bdc3c6');
                                                  byteawithoutorderwithequalcolin
-------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------

\x01c35301bf421c8edf38c34704bcc82838742917778ccb402a1b7452ad4a6ac7371acc0ac33100000035fe3424
919854c86194f1aa5bb4e1ca656e8fc6d05324a1419b69f488bdc3c6
(1 row)
```

Implementations such as byteawithoutorderwithequalcolin search for CEK and determine whether it is a normal encrypted data type.

Therefore, if the format of the input data is not the encrypted data format and the corresponding CEK does not exist on the local host, an error is returned.

```
gaussdb=# SELECT * FROM
byteawithoutorderwithequalcolsend('\x907219912381298461289346129'::byteawithoutorderwithequalcol);
ERROR:  cek with OID 596711794 not found
LINE 1: SELECT * FROM byteawithoutorderwithequalcolsend('\x907219912...
                                                        ^
gaussdb=# SELECT * FROM byteawithoutordercolout('\x907219019999999999999123812984612893461 29');
ERROR:  cek with OID 2566986098 not found
LINE 1: SELECT * FROM byteawithoutordercolout('\x9072190199999999999...

gaussdb=# SELECT * FROM
byteawithoutorderwithequalcolrecv('\x907219019999999999999123812984612893461 29'::byteawithoutorde
rwithequalcol);
ERROR:  cek with OID 2566986098 not found
                                      ^
gaussdb=# SELECT * FROM
byteawithoutorderwithequalcolsend('\x907219019999999999999123812984612893461 29'::byteawithoutorde
rwithequalcol);
ERROR:  cek with OID 2566986098 not found
LINE 1: SELECT * FROM byteawithoutorderwithequalcolsend('\x907219019...
                                                        ^
```

7.6.23 Set Returning Functions

Series Generating Functions

- generate_series(start, stop)

  Description: Generates a series of values, from **start** to **stop** with a step size of one.

  Parameter type: int, bigint, numeric

  Return type: setof int, setof bigint, setof numeric (same as the parameter type)

- generate_series(start, stop, step)

  Description: Generates a series of values, from **start** to **stop** with a step size of **step**.

  Parameter type: int, bigint, numeric

  Return type: setof int, setof bigint, setof numeric (same as the parameter type)

- generate_series(start, stop, step interval)

  Description: Generates a series of values, from **start** to **stop** with a step size of **step**.

  Parameter type: timestamp or timestamp with time zone

  Return type: setof timestamp or setof timestamp with time zone (same as parameter type)

When **step** is positive, zero rows are returned if **start** is greater than **stop**. Conversely, when **step** is negative, zero rows are returned if **start** is less than **stop**. Zero rows are also returned for **NULL** inputs. It is an error for **step** to be zero.

Example:

```
gaussdb=# SELECT * FROM generate_series(2,4);
 generate_series
-----------------
               2
               3
               4
(3 rows)

gaussdb=# SELECT * FROM generate_series(5,1,-2);
 generate_series
-----------------
               5
               3
               1
(3 rows)

gaussdb=# SELECT * FROM generate_series(4,3);
 generate_series
-----------------
(0 rows)

-- This example applies to the date-plus-integer operator.
gaussdb=# SELECT current_date + s.a AS dates FROM generate_series(0,14,7) AS s(a);
   dates
------------
 2017-06-02
 2017-06-09
 2017-06-16
(3 rows)

gaussdb=# SELECT * FROM generate_series('2008-03-01 00:00'::timestamp, '2008-03-04 12:00', '10 hours');
   generate_series
---------------------
 2008-03-01 00:00:00
 2008-03-01 10:00:00
 2008-03-01 20:00:00
 2008-03-02 06:00:00
 2008-03-02 16:00:00
 2008-03-03 02:00:00
 2008-03-03 12:00:00
 2008-03-03 22:00:00
 2008-03-04 08:00:00
(9 rows)
```

- gs_search_function_with_name(funcname)

  Description: Obtains the OID of the funcname function.

  Parameter type: cstring

  Return type: SETOF oid

  ```
  -- Basic usage
  CREATE OR REPLACE FUNCTION proc_plpgsql(a int,b int, c int)
  RETURNS int AS $BODY$
  DECLARE
  BEGIN
  RETURN $1 + $2;
  END;
  $BODY$ language plpgsql;
  CREATE OR REPLACE FUNCTION proc_plpgsql(int,int)
  RETURNS int AS $BODY$
  DECLARE
  BEGIN
  RETURN $1 + $2;
  ```



```
END;
$BODY$ language plpgsql;
gaussdb=# SELECT gs_search_function_with_name('proc_plpgsql');
 gs_search_function_with_name
------------------------------
                        16776
                        24576
(2 rows)
```

Subscript Generating Functions

- generate_subscripts(array anyarray, dim int)

  Description: Generates a series comprising the given array's subscripts.

  Return type: setof int

- generate_subscripts(array anyarray, dim int, reverse boolean)

  Description: Generates a series comprising the given array's subscripts. When **reverse** is true, the series is returned in reverse order.

  Return type: setof int

generate_subscripts is a function that generates the set of valid subscripts for the specified dimension of the given array. Zero rows are returned for arrays that do not have the requested dimension, or for NULL arrays (but valid subscripts are returned for NULL array elements). Example:

```
-- Basic usage
gaussdb=# SELECT generate_subscripts('{NULL,1,NULL,2}'::int[], 1) AS s;
 s
---
 1
 2
 3
 4
(4 rows)
-- Unnest a 2D array.
gaussdb=# CREATE OR REPLACE FUNCTION unnest2(anyarray)
RETURNS SETOF anyelement AS $$
SELECT $1[i][j]
   FROM generate_subscripts($1,1) g1(i),
        generate_subscripts($1,2) g2(j);
$$ LANGUAGE sql IMMUTABLE;

gaussdb=# SELECT * FROM unnest2(ARRAY[[1,2],[3,4]]);
 unnest2
---------
       1
       2
       3
       4
(4 rows)

-- Delete the function.
gaussdb=# DROP FUNCTION unnest2;
```

7.6.24 Conditional Expression Functions

Conditional Expression Functions

- coalesce(expr1, expr2, ..., exprn)

  Description:

  Returns the first of its arguments that are not null.

COALESCE(expr1, expr2) is equivalent to **CASE WHEN expr1 IS NOT NULL THEN expr1 ELSE expr2 END**.

Example:

```
gaussdb=# SELECT coalesce(NULL,'hello');
 coalesce
----------
 hello
(1 row)
```

Note:

- Null is returned only if all parameters are null.
- It is often used to replace **NULL** with the default value.
- Like a CASE expression, COALESCE only evaluates the parameters that are needed to determine the result. That is, parameters to the right of the first non-NULL parameter are not evaluated.

- decode(base_expr, compare1, value1, Compare2,value2, ... default)

  Description: Compares **base_expr** with each **compare(n)** and **returns value(n)** if they are matched. If no matching result is found, the default value is returned.

  Example:

  ```
  gaussdb=# SELECT decode('A','A',1,'B',2,0);
   case
  ------
   1
  (1 row)
  ```

  Note:

  - Operations on the XML data are not supported.

- nullif(expr1, expr2)

  Description: Returns **NULL** only when **expr1** is equal to **expr2**. Otherwise, **expr1** is returned.

  nullif(expr1, expr2) is equivalent to **CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END**.

  NOTE

  The nullif(expr1, expr2) function is a mapping function. Therefore, the corresponding function definition cannot be found in the pg_proc system catalog.

  Example:

  ```
  gaussdb=# SELECT nullif('hello','world');
   nullif
  --------
   hello
  (1 row)
  ```

  Note:

  - Operations on the XML data are not supported.
  - Assume the two parameter data types are different:

    - If implicit conversion exists between the two data types, implicitly convert the parameter of lower priority to this data type using the data type of higher priority. If the conversion succeeds, computation is performed. Otherwise, an error is reported. Example:

      ```
      gaussdb=# SELECT nullif('1234'::VARCHAR,123::INT4);
       nullif
      --------
      ```



```
  1234
(1 row)
gaussdb=# SELECT nullif('1234'::VARCHAR,'2012-12-24'::DATE);
ERROR:  invalid input syntax for type timestamp: "1234"
```

  - If implicit conversion is not applied between two data types, an error is displayed. Example:

```
gaussdb=# SELECT nullif(1::bit, '1'::MONEY);
ERROR:  operator does not exist: bit = money
LINE 1: SELECT nullif(1::bit, '1'::MONEY);
                      ^
HINT:  No operator matches the given name and argument type(s). You might need to add explicit type casts.
CONTEXT:  referenced column: nullif
```

- nvl(expr1 , expr2)

  Description:

  – If the value of **expr1** is **NULL**, the value of **expr2** is returned.

  – If the value of **expr1** is not **NULL**, the value of **expr1** is returned.

  Example:

```
gaussdb=# SELECT nvl('hello','world');
 nvl
-------
 hello
(1 row)
```

  Note: Parameters **expr1** and **expr2** can be of any data type. If **expr1** and **expr2** are of different data types, NVL checks whether **expr2** can be implicitly converted to **expr1**. If it can, the data type of **expr1** is returned. Otherwise, an error is returned.

- nvl2(expr1, expr2, expr3)

  Description:

  – If *expr1* is **NULL**, *expr3* is returned.

  – If *expr1* is not **NULL**, *expr2* is returned.

  > NOTE
  >
  > This function is valid only when **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1**.

  Example:

```
gaussdb=# SELECT nvl2('hello','world','other');
 case
-------
 world
(1 row)
```

  Note: The *expr2* and *expr3* parameters can be of any type. If the last two parameters of NVL2 are of different types, check whether *expr3* can be implicitly converted to *expr2*. If *expr3* cannot be implicitly converted to *expr2*, an error is returned. If the first parameter is of the numeric type, the function converts this parameter and other parameters to the numeric type, and then compares them. If the parameters cannot be converted, an error message is displayed. If the first parameter is of another type, the function converts other parameters to the type of the first parameter for comparison. If the parameters cannot be converted, an error message is displayed.

- greatest(expr1 [, ...])

  Description: Selects the largest value from a list of any number of expressions.

Return type:

Example:

```
gaussdb=# SELECT greatest(1*2,2-3,4-1);
 greatest
----------
        3
(1 row)
gaussdb=# SELECT greatest('HARRY', 'HARRIOT', 'HAROLD');
 greatest
----------
 HARRY
(1 row)
```

Note:

Operations on the XML data are not supported.

NOTE

When **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1**:

- If the value of any parameter is **NULL**, the function returns **NULL**.
- If the first parameter is of the numeric type, the function converts this parameter and other parameters to the numeric type, and then compares them. If the parameters cannot be converted, an error message is displayed. If the first parameter is of another type, the function converts other parameters to the type of the first parameter for comparison. If the parameters cannot be converted, an error message is displayed.

- least(expr1 [, ...])

  Description: Selects the smallest value from a list of any number of expressions.

  Example:

  ```
  gaussdb=# SELECT least(1*2,2-3,4-1);
   least
  -------
     -1
  (1 row)
  gaussdb=# SELECT least('HARRY','HARRIOT','HAROLD');
   least
  --------
   HAROLD
  (1 row)
  ```

  Note:

  Operations on the XML data are not supported.

  NOTE

  When **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1**:

  - If the value of any parameter is **NULL**, the function returns **NULL**.
  - If the first parameter is of the numeric type, the function converts this parameter and other parameters to the numeric type, and then compares them. If the parameters cannot be converted, an error message is displayed. If the first parameter is of another type, the function converts other parameters to the type of the first parameter for comparison. If the parameters cannot be converted, an error message is displayed.

- EMPTY_BLOB()

  Description: Initiates a BLOB variable in an INSERT or an UPDATE statement to a **NULL** value.

  Return type: BLOB

Example:

```
-- Create a table.
gaussdb=# CREATE TABLE blob_tb(b blob,id int);
-- Insert data.
gaussdb=# INSERT INTO blob_tb VALUES (empty_blob(),1);
-- Drop the table.
gaussdb=# DROP TABLE blob_tb;
```

Note: The length is 0 obtained using **DBE_LOB.GET_LENGTH**.

- EMPTY_CLOB()

  Description: Initiates a CLOB variable in an INSERT or UPDATE statement to a null value.

  NOTE

  This function is valid only when **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1**.

  Return type: CLOB

  Example:

```
-- Create a table.
gaussdb=# CREATE TABLE clob_tb(c clob,id int);
-- Insert data.
gaussdb=# INSERT INTO clob_tb VALUES (empty_clob(),1);
-- Drop the table.
gaussdb=# DROP TABLE clob_tb;
```

  Note: The length is 0 obtained using **DBE_LOB.GET_LENGTH**.

- lnnvl(condition)

  Description: Checks the condition in the WHERE clause of a query statement. If the condition is true, **false** is returned. If the condition is unknown or false, **true** is returned.

  condition: The value must be a logical expression but cannot be used in composite conditions with keywords such as AND, OR, and BETWEEN.

  Return type: Boolean

  Example:

```
-- Create a table.
gaussdb=# CREATE TABLE student_demo (name VARCHAR2(20), grade  NUMBER(10,2));
CREATE TABLE

-- Insert data.
gaussdb=# INSERT INTO student_demo VALUES ('name0',0);
INSERT 0 1
gaussdb=# INSERT INTO student_demo VALUES ('name1',1);
INSERT 0 1
gaussdb=# INSERT INTO student_demo VALUES ('name2',2);
INSERT 0 1

-- Invoke lnnvl.
gaussdb=# SELECT * FROM student_demo WHERE LNNVL(name = 'name1');
 name  | grade
-------+-------
 name0 |  0.00
 name2 |  2.00
(2 rows)
```

  NOTE

  This function is valid only when **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s2**.

- isnull(expr)

  Description: Checks whether **expr** is **NULL**. If **expr** is **NULL**, **true** is returned. Otherwise, **false** is returned. The isnull function is the mapping of the IS NULL expression. isnull(expr) is equivalent to the expr is null expression. The isnull function is compatible in full mode.

  Parameter: Any type of input is supported.

  Return type: Boolean

  Example:

```
gaussdb=# SELECT isnull(null);
 ?column?
----------
 t
(1 row)

gaussdb=# SELECT isnull(1);
 ?column?
----------
 f
(1 row)
```

  NOTE

  The isnull(expr) function is a mapping function. Therefore, the corresponding function definition cannot be found in the pg_proc system catalog.

- if(expr1, expr2, expr3)

  Description: Checks the value of **expr1**. If **expr1** is **true**, **expr2** is returned. Otherwise, **expr3** is returned.

  if(expr1, expr2, expr3) is logically equivalent to CASE WHEN expr1 THEN expr2 ELSE expr3 END.

  Parameters: See **Table 7-78**.

  Table 7-78 Parameter types

Parameter	Valid Input Parameter Type	Description
expr1	BOOLEAN	Determines the return value of the if() function.
expr2	Any type	If **expr1** is **true**, **expr2** is returned.
expr3	Any type	If **expr1** is **false**, **expr3** is returned.

  Return value type: related to the input parameter type. For details about the output rules, see the description.

NOTE

- if(expr1, expr2, expr3) is supported only in B compatibility mode.
- When the compatibility parameters **b_format_version** is set to **''** and **b_format_dev_version** is set to **''**, the output result type of the if(expr1, expr2, expr3) function is the same as that of the CASE WHEN expr1 THEN expr2 ELSE expr3 END function. When the compatibility parameters **b_format_version** is set to **'5.7'** and **b_format_dev_version** is set to **'s1'**, the return value type is deduced according to the following rules in descending order:
  - If the two input parameters are of the same type, the return value type is the same as the input parameter type.
  - If one of the input parameters is of the SET type, the return value is of the TEXT type.
  - If one of the input parameters is of the BLOB type, the return value is of the BLOB type.
  - If one of the input parameters is of the TEXT type, the return value is of the TEXT type.
  - If one of the input parameters is of the STRING type, the return value is of the TEXT type.
  - If the two input parameters are of the time type, the return value is of the TIMESTAMPTZ type; if one of the input parameters is of the time type but the other is not, the return value is of the TEXT type.
  - If both input parameters are signed integers and one input parameter is of the BIGINT type, the return value is of the BIGINT type; otherwise, the return value is of the INT type.
  - If both input parameters are unsigned integers and one input parameter is of the UNSIGNED BIGINT type, the return value is of the UNSIGNED BIGINT type; otherwise, the return value is of the UNSIGNED INT type.
  - If the two input parameters are of the floating point type and integer type respectively, the return value is of the floating point type.
  - Except the preceding NUMERIC type combinations, the return value for other NUMERIC type combinations is of the NUMERIC type.
  - If no implicit conversion function exists between the input parameter type and return value type, an error is reported.
- If the input parameter **expr1** is **NULL**, the returned result is the same as that when **expr1** is **FALSE**.
- The if(expr1, expr2, expr3) function is a mapping function. Therefore, the corresponding function definition cannot be found in the pg_proc system catalog.

Example:

```
-- expr1 is an expression.
gaussdb=# SELECT if(2>3, 'true', 'false');
 case
-------
 false
(1 row)

-- If the input parameter expr1 is NULL, the result is the same as that when expr1 is false, and the value of expr2 is returned.
gaussdb=# SELECT if(null, 'not null', 'is null');
  case
---------
 is null
(1 row)
```

- ifnull(expr1, expr2)

  Description: Returns the value of **expr2** if **expr1** is **NULL**; otherwise, returns the value of **expr1**.

ifnull(expr1, expr2) is logically equivalent to nvl(expr1, expr2).

Parameter: any type.

Return value type: related to the input parameter type. For details about the output rules, see the description.

NOTE

- ifnull(expr1, expr2) is supported only in B compatibility mode.
- When the compatibility parameters **b_format_version** is set to **''** and **b_format_dev_version** is set to **''**, the output result type of the ifnull(expr1, expr2) function is the same as that of the nvl(expr1, expr2) function. When the compatibility parameters **b_format_version** is set to **'5.7'** and **b_format_dev_version** is set to **'s1'**, the return value type is deduced according to the following rules in descending order:
  - If the two input parameters are of the same type, the return value type is the same as the input parameter type.
  - If one of the input parameters is of the SET type, the return value is of the TEXT type.
  - If one of the input parameters is of the BLOB type, the return value is of the BLOB type.
  - If one of the input parameters is of the TEXT type, the return value is of the TEXT type.
  - If one of the input parameters is of the STRING type, the return value is of the TEXT type.
  - If the two input parameters are of the time type, the return value is of the TIMESTAMPTZ type; if one of the input parameters is of the time type but the other is not, the return value is of the TEXT type.
  - If both input parameters are of the NUMERIC type, the return value is of the type with higher precision. For example, if the input parameter types are TINYINT and INT, the return value type is INT.
  - If no implicit conversion function exists between the input parameter type and return value type, an error is reported.
- If both input parameters are **NULL**, the return value is **NULL**; otherwise, the first value that is not **NULL** is returned.
- The ifnull(expr1, expr2) function is a mapping function. Therefore, the corresponding function definition cannot be found in the pg_proc system catalog.

Example:

```
-- If the input parameter is an empty string, the output is an empty string instead of NULL.
gaussdb=# SELECT ifnull('', null) is null as a;
 a
---
 f
(1 row)

-- If both input parameters are NULL, the output is NULL.
gaussdb=# SELECT ifnull(null, null) is null as a;
 a
---
 t
(1 row)

-- If the input parameters are NULL and a character string, the output is the first non-null value.
gaussdb=# SELECT ifnull(null, 'A') as a;
 a
---
 A
(1 row)
```


7.6.25 System Information Functions

Session Information Functions

- SYS_CONTEXT()

  Description: Returns the value of the parameter associated with the context namespace at the current time.

  Return type: text

  Example:

```
SELECT SYS_CONTEXT('userenv','NLS_CURRENCY');
 sys_context
-------------
 $
(1 row)

SELECT SYS_CONTEXT('userenv','NLS_DATE_FORMAT');
  sys_context
---------------
 ISO, MDY
(1 row)

SELECT SYS_CONTEXT('userenv','NLS_DATE_LANGUAGE');
 sys_context
-------------
 en_US.UTF-8
(1 row)
```

- current_catalog

  Description: Name of the current database (called "catalog" in the SQL standard)

  Return type: name

  Example:

```
testdb=# SELECT current_catalog;
 current_database
------------------
 testdb
(1 row)
```

- current_database()

  Description: Name of the current database

  Return type: name

  Example:

```
testdb=# SELECT current_database();
 current_database
------------------
 testdb
(1 row)
```

- current_query()

  Description: Text of the current query execution committed by the client (which might contain more than one statement)

  Return type: text

  Example:

```
gaussdb=# SELECT current_query();
      current_query
--------------------------
 SELECT current_query();
(1 row)
```


- current_schema[()]

  Description: Name of the current schema

  Return type: name

  Example:

  ```
  gaussdb=# SELECT current_schema();
   current_schema
  ----------------
   public
  (1 row)
  ```

  Note: **current_schema** returns the first valid schema name in the search path. (If the search path is empty or contains no valid schema name, **NULL** is returned.) This is the schema that will be used for any tables or other named objects that are created without specifying a target schema.

- current_schemas(Boolean)

  Description: Name of a schema in the search path

  Return type: name[]

  Example:

  ```
  gaussdb=# SELECT current_schemas(true);
    current_schemas
  ---------------------
   {pg_catalog,public}
  (1 row)
  ```

  Note:

  current_schemas(Boolean) returns an array of the names of all schemas in the search path. The Boolean option specifies whether implicitly included system schemas such as **pg_catalog** are included in the returned search path.

  NOTE

  The search path can be altered at run time. The command is as follows:

  ```
  SET search_path TO schema [, schema, ...]
  ```

- database()

  Description: Returns the name of the current schema.

  Parameter: none

  Return type: name

  Example:

  ```
  gaussdb=# SELECT database();
   database
  ----------
   public
  (1 row)
  ```

  NOTE

  This function takes effect only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**.

- current_user

  Description: Username in the current execution environment.

  Return type: name

  Example:

  ```
  gaussdb=# SELECT current_user;
   current_user
  ```



```
--------------
 omm
(1 row)
```

Note: **current_user** is the user identifier used for permission check. Normally it is equal to the session user, but it can be changed by using **SET ROLE**. It also changes during the execution of functions with the **SECURITY DEFINER** attribute.

- definer_current_user

  Description: Username in the current execution environment.

  Return type: name

  Example:

```
gaussdb=# SELECT definer_current_user();
 definer_current_user
----------------------
 omm
(1 row)
```

- pg_current_sessionid()

  Description: Session ID in the current execution environment.

  Return type: text

  Example:

```
gaussdb=# SELECT pg_current_sessionid();
    pg_current_sessionid
----------------------------
 1579228402.140190434944768
(1 row)
```

  Note: **pg_current_sessionid()** is used to obtain the session ID in the current execution environment. The structure of the value is *Timestamp.Session ID*. When **enable_thread_pool** is set to **on**, the actual session ID is used. When **enable_thread_pool** is set to **off**, the session ID is the thread ID.

- pg_current_sessid

  Description: Session ID in the current execution environment.

  Return type: text

  Example:

```
gaussdb=# select pg_current_sessid();
pg_current_sessid
-------------------
140308875015936
(1 row)
```

  Note: In thread pool mode, the ID of the current session is obtained. In non-thread pool mode, the backend thread ID of the current session is obtained.

- pg_current_userid

  Description: Current user ID.

  Return type: text

```
gaussdb=# SELECT pg_current_userid();
pg_current_userid
-------------------
10
(1 row)
```

- working_version_num()

  Description: Version number. It returns a version number related to system compatibility.

Return type: int

Example:

```
gaussdb=# SELECT working_version_num();
 working_version_num
---------------------
               92231
(1 row)
```

- tablespace_oid_name(oid)

  Description: Queries the tablespace name based on the tablespace OID.

  Return type: text

  Example:

  ```
  gaussdb=# SELECT tablespace_oid_name(1663);
   tablespace_oid_name
  ---------------------
   pg_default
  (1 row)
  ```

- inet_client_addr()

  Description: Remote connection address. **inet_client_addr** returns the IP address of the current client.

  NOTE

  This function is valid only in remote connection mode.

  Return type: inet

  Example:

  ```
  gaussdb=# SELECT inet_client_addr();
   inet_client_addr
  ------------------
   10.10.0.50
  (1 row)
  ```

- inet_client_port()

  Description: Remote connection port. And **inet_client_port** returns the port number of the current client.

  NOTE

  This function is valid only in remote connection mode.

  Return type: int

  Example:

  ```
  gaussdb=# SELECT inet_client_port();
   inet_client_port
  ------------------
             33143
  (1 row)
  ```

- inet_server_addr()

  Description: Local connection address. **inet_server_addr** returns the IP address on which the server accepts the current connection.

  NOTE

  This function is valid only in remote connection mode.

  Return type: inet

  Example:

```
gaussdb=# SELECT inet_server_addr();
 inet_server_addr
------------------
 10.10.0.13
(1 row)
```

- inet_server_port()

  Description: Local connection port. **inet_server_port** returns the number of the port receiving the current connection. All these functions return **NULL** if the current connection is via a Unix-domain socket.

  NOTE

  This function is valid only in remote connection mode.

  Return type: int

  Example:

```
gaussdb=# SELECT inet_server_port();
 inet_server_port
------------------
             8000
(1 row)
```

- pg_backend_pid()

  Description: Thread ID of the service thread connected to the current session.

  Return type: int

  Example:

```
gaussdb=# SELECT pg_backend_pid();
 pg_backend_pid
------------------
 140229352617744
(1 row)
```

- pg_conf_load_time()

  Description: Configures load time. **pg_conf_load_time** returns the timestamp when the server configuration files were last loaded.

  Return type: timestamp with time zone

  Example:

```
gaussdb=# SELECT pg_conf_load_time();
      pg_conf_load_time
-------------------------------
 2017-09-01 16:05:23.89868+08
(1 row)
```

- pg_my_temp_schema()

  Description: OID of the temporary schema of a session. The value is **0** if the OID does not exist.

  Return type: oid

  Example:

```
gaussdb=# SELECT pg_my_temp_schema();
 pg_my_temp_schema
-------------------
                 0
(1 row)
```

  Note: **pg_my_temp_schema** returns the OID of the current session's temporary schema, or **0** if it has no temporary schemas (because no temporary tables are created). **pg_is_other_temp_schema** returns **true** if the given OID is the OID of another session's temporary schema.

- pg_is_other_temp_schema(oid)

  Description: Specifies whether the schema is the temporary schema of another session.

  Return type: Boolean

  Example:

```
gaussdb=# SELECT pg_is_other_temp_schema(25356);
 pg_is_other_temp_schema
-------------------------
 f
(1 row)
```

- pg_listening_channels()

  Description: Name of the channel that the session is currently listening on.

  Return type: SETOF text

  Example:

```
gaussdb=# SELECT pg_listening_channels();
 pg_listening_channels
-----------------------
(0 rows)
```

  Note: **pg_listening_channels** returns a set of names of channels that the current session is currently listening on.

- pg_postmaster_start_time()

  Description: Server start time. **pg_postmaster_start_time** returns the timestamp with time zone when the server is started.

  Return type: timestamp with time zone

  Example:

```
gaussdb=# SELECT pg_postmaster_start_time();
   pg_postmaster_start_time
-------------------------------
 2017-08-30 16:02:54.99854+08
(1 row)
```

- pg_get_ruledef(rule_oid)

  Description: Obtains the **CREATE RULE** command for a rule.

  Return type: text

  Example:

```
gaussdb=# SELECT * FROM pg_get_ruledef(24828);
                              pg_get_ruledef
-----------------------------------------------------------------------
 CREATE RULE t1_ins AS ON INSERT TO t1 DO INSTEAD INSERT INTO t2 (id) VALUES (new.id);
(1 row)
```

- sessionid2pid()

  Description: Obtains PID information from a session ID (for example, the **sessid** column in **gs_session_stat**).

  Return type: int8

  Example:

```
gaussdb=# SELECT sessionid2pid(sessid::cstring) FROM gs_session_stat LIMIT 2;
  sessionid2pid
-----------------
 139973107902208
 139973107902208
(2 rows)
```


- session_context('namespace' , 'parameter')

  Description: Obtains and returns the parameter values of a specified namespace.

  Return type: VARCHAR

  Example:

  ```
  gaussdb=# SELECT session_context('USERENV', 'CURRENT_SCHEMA');
  session_context
  ----------------
   public
  (1 row)
  ```

  Note: Currently, the **current_user**, **current_schema**, **client_info**, **ip_address**, **sessionid**, and **sid** parameters are supported.

- pg_trigger_depth()

  Description: Nesting level of triggers.

  Return type: int

  Example:

  ```
  gaussdb=# SELECT pg_trigger_depth();
   pg_trigger_depth
  ------------------
                  0
  (1 row)
  ```

- session_user

  Description: Session username.

  Return type: name

  Example:

  ```
  gaussdb=# SELECT session_user;
   session_user
  --------------
   omm
  (1 row)
  ```

  Note: **session_user** usually specifies the initial user connected to the current database, but the system administrator can change this setting by using **SET SESSION AUTHORIZATION**.

- user

  Description: Equivalent to current_user.

  Return type: name

  Example:

  ```
  gaussdb=# SELECT user;
   current_user
  --------------
   omm
  (1 row)
  ```

- getpgusername()

  Description: Obtains the database username.

  Return type: name

  Example:

  ```
  gaussdb=# SELECT getpgusername();
   getpgusername
  ---------------
   GaussDB_userna
  (1 row)
  ```


- getdatabaseencoding()

  Description: Obtains the database encoding mode.

  Return type: name

  Example:

  ```
  gaussdb=# SELECT getdatabaseencoding();
   getdatabaseencoding
  ---------------------
   SQL_ASCII
  (1 row)
  ```

- version()

  Description: Version information. **version** returns a string describing a server's version.

  Return type: text

  Example:

  ```
  gaussdb=# SELECT version();
  version
  ----------------------------------------------------------------------------------------------------------------------------------------------------------------
  gaussdb (GaussDB Kernel XXX.X.XXX build fab4f5ea) compiled at 2021-10-24 11:58:22 commit 3086 last mr 6592 release
  (1 row)
  ```

- opengauss_version()

  Description: openGauss version information.

  Return type: text

  The following is an example. Replace *x.x.x* in the query result with the actual value.

  ```
  gaussdb=# SELECT opengauss_version();
   opengauss_version
  -------------------
   x.x.x
  (1 row)
  ```

- gs_deployment()

  Description: Information about the deployment mode of the current system.

  Return type: text

  Example:

  ```
  gaussdb=# SELECT gs_deployment();
     gs_deployment
  ---------------------
   BusinessCentralized
  (1 row)
  ```

- get_hostname()

  Description: Returns the host name of the current node.

  Return type: text

  Example:

  ```
  gaussdb=# SELECT get_hostname();
   get_hostname
  --------------
   linux-user
  (1 row)
  ```

- get_nodename()

  Description: Returns the name of the current node.

Return type: text

Example:

```
gaussdb=# SELECT get_nodename();
 get_nodename
--------------
 datanode1
(1 row)
```

- get_nodeinfo(text)

  Description: Returns the value of the corresponding node information based on the search attribute. Currently, the search attributes include **node_name** and **node_type**.

  Return type: text

  Example:

  ```
  gaussdb=# SELECT get_nodeinfo('node_type');
   get_nodeinfo
  --------------
   CEN_DN
  (1 row)
  gaussdb=# SELECT get_nodeinfo('node_name');
   get_nodeinfo
  --------------
   datanode1
  (1 row)
  ```

- get_schema_oid(cstring)

  Description: Returns the OID of the queried schema.

  Return type: oid

  Example:

  ```
  gaussdb=# SELECT get_schema_oid('public');
   get_schema_oid
  ----------------
             2200
  (1 row)
  ```

- get_client_info()

  Description: Returns client information.

  Return type: record

Access Permission Query Functions

Permissions on DDL statements, including ALTER, DROP, COMMENT, INDEX, and VACUUM, are inherent permissions implicitly owned by the owner.

The following access permission query function only indicates whether a user has a certain permission on an object. That is, the permission on the object recorded in the **acl** column of the system catalog is returned.

- has_any_column_privilege(user, table, privilege)

  Description: Queries whether a specified user has permission for any column of a table.

Table 7-79 Parameter type description

Parameter	Valid Input Parameter Type
user	name, oid
table	text, oid
privilege	text

Return type: Boolean

- has_any_column_privilege(table, privilege)

  Description: Checks whether the current user has the permission to access any column of a table. For details about the valid parameter types, see **Table 7-79**.

  Return type: Boolean

  Note: **has_any_column_privilege** checks whether a user can access any column of a table in a particular way. Its parameter possibilities are analogous to those of **has_table_privilege**, except that the desired access permission type must be some combination of SELECT, INSERT, UPDATE, COMMENT or REFERENCES.

  NOTE

  Note that having any of these permissions at the table level indicates that the permission is implicitly granted for each column of the table. Therefore, **has_any_column_privilege** always returns **true** if **has_table_privilege** has the same parameters. A success message is also returned if a column-level permission is granted for at least one column.

- has_column_privilege(user, table, column, privilege)

  Description: Specifies whether a specified user has the permission to access columns.

Table 7-80 Parameter type description

Parameter	Valid Input Parameter Type
user	name, oid
table	text, oid
column	text, smallint
privilege	text

Return type: Boolean

- has_column_privilege(table, column, privilege)

  Description: Checks whether the current user has the permission to access any column. For details about the valid parameter types, see **Table 7-80**.

  Return type: Boolean

has_column_privilege checks whether a user can access a column in a particular way. Its parameter possibilities are analogous to those of **has_table_privilege**, with the addition that the column can be specified either by name or attribute number. The desired access permission type must be some combination of **SELECT**, **INSERT**, **UPDATE**, **COMMENT** or **REFERENCES**.

NOTE

Note that having any of these permissions at the table level indicates that the permission is implicitly granted for each column of the table.

- has_cek_privilege(user, cek, privilege)

  Description: Specifies whether a specified user has permission for CEKs. The following table describes the parameters.

  Table 7-81 Parameter type description

Parameter	Valid Input Parameter Type	Description	Value Range
user	name, oid	A specific user	Username or ID
cek	text, oid	CEK	Name or ID of a CEK
privilege	text	Permission	• **USAGE**: allows users to use the specified CEK. • **DROP**: allows users to delete the specified CEK.

  Return type: Boolean

- has_cmk_privilege(user, cmk, privilege)

  Description: Specifies whether a specified user has permission for CMKs. The following table describes the parameters.

  Table 7-82 Parameter type description

Parameter	Valid Input Parameter Type	Description	Value Range
user	name, oid	A specific user	Username or ID
cmk	text, oid	CMK	Name or ID of the CMK

Parameter	Valid Input Parameter Type	Description	Value Range
privilege	text	Permission	• **USAGE**: allows users to use the specified CMK. • **DROP**: allows users to delete the specified CMK.

Return type: Boolean

- has_database_privilege(user, database, privilege)

  Description: Specifies whether a specified user has permission for databases. The following table describes the parameters.

  Table 7-83 Parameter type description

Parameter	Valid Input Parameter Type
user	name, oid
database	text, oid
privilege	text

  Return type: Boolean

- has_database_privilege(database, privilege)

  Description: Checks whether the current user has permission to access a database. For details about the valid parameter types, see **Table 7-83**.

  Return type: Boolean

  Note: **has_database_privilege** checks whether a user can access a database in a particular way. Its parameter possibilities are analogous to those of **has_table_privilege**. The desired access permission type must be some combination of **CREATE**, **CONNECT**, **TEMPORARY**, **ALTER**, **DROP**, **COMMENT** or **TEMP** (which is equivalent to **TEMPORARY**).

- has_directory_privilege(user, directory, privilege)

  Description: Specifies whether a specified user has permission for directories.

  Table 7-84 Parameter type description

Parameter	Valid Input Parameter Type
user	name, oid
directory	text, oid
privilege	text

Return type: Boolean

- has_directory_privilege(directory, privilege)

  Description: Checks whether the current user has the permission to access the directory. For details about the valid parameter types, see **Table 7-84**.

  Return type: Boolean

- has_foreign_data_wrapper_privilege(user, fdw, privilege)

  Description: Specifies whether a specified user has permission for foreign data wrappers.

  Table 7-85 Parameter type description

Parameter	Valid Input Parameter Type
user	name, oid
fdw	text, oid
privilege	text

  Return type: Boolean

- has_foreign_data_wrapper_privilege(fdw, privilege)

  Description: Specifies whether the current user has permission for accessing foreign-data wrappers. For details about the valid parameter types, see **Table 7-85**.

  Return type: Boolean

  Note: **has_foreign_data_wrapper_privilege** checks whether a user can access a foreign data wrapper in a particular way. Its parameter possibilities are analogous to those of **has_table_privilege**. The desired access permission type must be **USAGE**.

- has_function_privilege(user, function, privilege)

  Description: Specifies whether a specified user has permission for functions.

  Table 7-86 Parameter type description

Parameter	Valid Input Parameter Type
user	name, oid
function	text, oid
privilege	text

  Return type: Boolean

- has_function_privilege(function, privilege)

  Description: Specifies whether the current user has permissions on functions. **Table 7-86** describes the valid parameter types.

  Return type: Boolean

Note: **has_function_privilege** checks whether a user can access a function in a particular way. Its parameter possibilities are analogous to those of **has_table_privilege**. When a function is specified by a text string rather than by an OID, the allowed input is the same as that for the regprocedure data type (see **Object Identifier Types**). The access permission type must be **EXECUTE**, **ALTER**, **DROP**, or **COMMENT**.

- has_language_privilege(user, language, privilege)

  Description: Specifies whether a specified user has permission for languages.

  Table 7-87 Parameter type description

Parameter	Valid Input Parameter Type
user	name, oid
language	text, oid
privilege	text

  Return type: Boolean

- has_language_privilege(language, privilege)

  Description: Specifies whether the current user has permissions on languages. **Table 7-87** describes the valid parameter types.

  Return type: Boolean

  Note: **has_language_privilege** checks whether a user can access a procedural language in a particular way. Its parameter possibilities are analogous to those of **has_table_privilege**. The desired access permission type must be **USAGE**.

- has_nodegroup_privilege(user, nodegroup, privilege)

  Description: Checks whether a user has permission to access a database node.

  Return type: Boolean

  Table 7-88 Parameter type description

Parameter	Valid Input Parameter Type
user	name, oid
nodegroup	text, oid
privilege	text

- has_nodegroup_privilege(nodegroup, privilege)

  Description: Checks whether a user has permission to access a database node. Its parameter possibilities are analogous to those of **has_table_privilege**. The access permission type must be **USAGE**, **CREATE**, **COMPUTE**, **ALTER**, or **DROP**.

  Return type: Boolean

- has_schema_privilege(user, schema, privilege)

  Description: Specifies whether a specified user has permission for schemas.

  Return type: Boolean

- has_schema_privilege(schema, privilege)

  Description: Specifies whether the current user has permission for schemas.

  Return type: Boolean

  Note: **has_schema_privilege** checks whether a user can access a schema in a particular way. Its parameter possibilities are analogous to those of **has_table_privilege**. The desired access permission type must be a combination among CREATE, USAGE, ALTER, DROP, and COMMENT. If the check type contains the CREATE permission and the checked schema is a schema with the same name as the user, the function returns **TRUE** only when the user has the owner permission for the schema due to the special constraint of the schema with the same name.

- has_server_privilege(user, server, privilege)

  Description: Specifies whether a specified user has permission for foreign servers.

  Return type: Boolean

- has_server_privilege(server, privilege)

  Description: Specifies whether the current user has permission for foreign servers.

  Return type: Boolean

  Note: **has_server_privilege** checks whether a user can access a foreign server in a particular way. Its parameter possibilities are analogous to those of **has_table_privilege**. The access permission type must be **USAGE**, **ALTER**, **DROP**, or **COMMENT**.

- has_table_privilege(user, table, privilege)

  Description: Specifies whether a specified user has permission for tables.

  Return type: Boolean

- has_table_privilege(table, privilege)

  Description: Specifies whether the current user has permission for tables.

  Return type: Boolean

  Note: **has_table_privilege** checks whether a user can access a table in a particular way. The user can be specified by name or by OID (**pg_authid.oid**), or be set to **public**, which indicates public pseudo roles. If this parameter is omitted, **current_user** is used. The table can be specified by name or by OID. When it is specified by name, the name can be schema-qualified if necessary. The desired access permission type is specified by a text string, which must be one of the values **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **TRUNCATE**, **REFERENCES**, **TRIGGER**, **ALTER**, **DROP**, **COMMENT**, **INDEX** or **VACUUM**. Optionally, **WITH GRANT OPTION** can be added to a permission type to test whether the permission is held with the grant option. Also, multiple permission types can be separated by commas (,), in which case the result will be **true** if any of the listed permissions is held.

  Example:

```
gaussdb=# SELECT has_table_privilege('tpcds.web_site', 'select');
 has_table_privilege
```



```
---------------------
 t
(1 row)

gaussdb=# SELECT has_table_privilege('omm', 'tpcds.web_site', 'select,INSERT WITH GRANT OPTION ');
 has_table_privilege
---------------------
 t
(1 row)
```

- has_tablespace_privilege(user, tablespace, privilege)

  Description: Specifies whether a specified user has permission for tablespaces.

  Return type: Boolean

- has_tablespace_privilege(tablespace, privilege)

  Description: Specifies whether the current user has permission for tablespaces.

  Return type: Boolean

  Note: **has_tablespace_privilege** checks whether a user can access a tablespace in a particular way. Its parameter possibilities are analogous to those of **has_table_privilege**. The access permission type must be **CREATE**, **ALTER**, **DROP**, or **COMMENT**.

- pg_has_role(user, role, privilege)

  Description: Specifies whether a specified user has permission for roles.

  Return type: Boolean

- pg_has_role(role, privilege)

  Description: Specifies whether the current user has permission for roles.

  Return type: Boolean

  Note: **pg_has_role** checks whether a user can access a role in a particular way. Its parameter possibilities are analogous to those of **has_table_privilege**, except that **public** cannot be used as a username. The desired access permission type must be some combination of **MEMBER** or **USAGE**. **MEMBER** denotes direct or indirect membership in the role (that is, permission **SET ROLE**), while **USAGE** denotes the usage permission on the role that is available without the **SET ROLE** permission.

- has_any_privilege(user, privilege)

  Description: Queries whether a specified user has certain ANY permission. If multiple permissions are queried at the same time, **true** is returned as long as one permission is obtained.

  Return type: Boolean

Table 7-89 Parameter type description

Parameter	Valid Input Parameter Type	Description	Value Range
user	name	A specific user	Existing username

Parameter	Valid Input Parameter Type	Description	Value Range
privilege	text	ANY permission	The options are as follows: • CREATE ANY TABLE [WITH ADMIN OPTION] • ALTER ANY TABLE [WITH ADMIN OPTION] • DROP ANY TABLE [WITH ADMIN OPTION] • SELECT ANY TABLE [WITH ADMIN OPTION] • INSERT ANY TABLE [WITH ADMIN OPTION] • UPDATE ANY TABLE [WITH ADMIN OPTION] • DELETE ANY TABLE [WITH ADMIN OPTION] • CREATE ANY SEQUENCE [WITH ADMIN OPTION] • CREATE ANY INDEX [WITH ADMIN OPTION] • CREATE ANY FUNCTION [WITH ADMIN OPTION] • EXECUTE ANY FUNCTION [WITH ADMIN OPTION] • CREATE ANY PACKAGE [WITH ADMIN OPTION] • EXECUTE ANY PACKAGE [WITH ADMIN OPTION] • CREATE ANY TYPE [WITH ADMIN OPTION]

Schema Visibility Query Functions

Each function performs the visibility check on one type of database objects. For functions and operators, an object in the search path is visible if there is no object of the same name and parameter data type earlier in the path. For operator classes, both name and associated index access methods are considered.

All these functions require OIDs to identify the objects to be checked. If you want to test an object by name, it is convenient to use the OID alias type (regclass, regtype, regprocedure, regoperator, regconfig, or regdictionary).

For example, a table is said to be visible if the schema where the table is located is in the search path and no table of the same name appears earlier in the search

path. This is equivalent to the statement that the table can be referenced by name without explicit schema qualification. To list the names of all visible tables, run the following command:

```
gaussdb=# SELECT relname FROM pg_class WHERE pg_table_is_visible(oid);
```

- pg_collation_is_visible(collation_oid)

  Description: Specifies whether the collation is visible in the search path.

  Return type: Boolean
- pg_conversion_is_visible(conversion_oid)

  Description: Specifies whether the conversion is visible in the search path.

  Return type: Boolean
- pg_function_is_visible(function_oid)

  Description: Specifies whether the function is visible in the search path.

  Return type: Boolean
- pg_opclass_is_visible(opclass_oid)

  Description: Specifies whether the operator class is visible in search path.

  Return type: Boolean
- pg_operator_is_visible(operator_oid)

  Description: Specifies whether the operator is visible in search path.

  Return type: Boolean
- pg_opfamily_is_visible(opclass_oid)

  Description: Specifies whether the operator family is visible in search path.

  Return type: Boolean
- pg_table_is_visible(table_oid)

  Description: Specifies whether the table is visible in the search path.

  Return type: Boolean
- pg_ts_config_is_visible(config_oid)

  Description: Specifies whether the text search configuration is visible in search path.

  Return type: Boolean
- pg_ts_dict_is_visible(dict_oid)

  Description: Specifies whether the text search dictionary is visible in search path.

  Return type: Boolean
- pg_ts_parser_is_visible(parser_oid)

  Description: Specifies whether the text search parser is visible in search path.

  Return type: Boolean
- pg_ts_template_is_visible(template_oid)

  Description: Specifies whether the text search template is visible in search path.

  Return type: Boolean
- pg_type_is_visible(type_oid)

  Description: Specifies whether the type (or domain) is visible in search path.

Return type: Boolean

System Catalog Information Functions

- format_type(type_oid, typemod)

  Description: Obtains the SQL name of a data type.

  Return type: text

  Note: **format_type** returns the SQL name of a data type based on the OID of the data type and possible modifiers. If the specific modifier is unknown, pass **NULL** at the position of the modifier. Modifiers are generally meaningful only for data types with length restrictions. The SQL name returned by **format_type** contains the length of the data type, which can be calculated by taking sizeof(int32) from actual storage length [actual storage length – sizeof(int32)] in the unit of bytes. 32-bit space is required to store the customized length set by users. Therefore, the actual storage length contains 4 bytes more than the customized length. In the following example, the SQL name returned by **format_type** is character varying(6), indicating the length of the varchar type is 6 bytes. Therefore, the actual storage length of the varchar type is 10 bytes.

  ```
  gaussdb=# SELECT format_type((SELECT oid FROM pg_type WHERE typname='varchar'), 10);
      format_type
  ----------------------
   character varying(6)
  (1 row)
  ```

- getdistributekey(table_name)

  Description: Obtains a distribution key for a hash table. Distribution is not supported in a standalone system and the return value of this function is empty.

- pg_check_authid(role_oid)

  Description: Checks whether a role name with given OID exists.

  Return type: Boolean

  Example:

  ```
  gaussdb=# SELECT pg_check_authid(1);
  pg_check_authid
  -----------------
  f
  (1 row)
  ```

- pg_describe_object(catalog_id, object_id, object_sub_id)

  Description: Obtains the description of a database object.

  Return type: text

  Note: pg_describe_object returns the description of a database object specified by a catalog OID, an object OID, and a (possibly zero) sub-object ID. This is useful to determine the identity of an object stored in the **pg_depend** catalog.

- pg_get_constraintdef(constraint_oid)

  Description: Obtains definition of a constraint.

  Return type: text

- pg_get_constraintdef(constraint_oid, pretty_bool)

  Description: Obtains definition of a constraint.

  Return type: text

Note: **pg_get_constraintdef** and **pg_get_indexdef** respectively reconstruct the creation command for a constraint and an index.

- pg_get_expr(pg_node_tree, relation_oid)

  Description: Decompiles internal form of an expression, assuming that any Vars in it refer to the relationship indicated by the second parameter.

  Return type: text

- pg_get_expr(pg_node_tree, relation_oid, pretty_bool)

  Description: Decompiles internal form of an expression, assuming that any Vars in it refer to the relationship indicated by the second parameter.

  Return type: text

  Note: **pg_get_expr** decompiles the internal form of an individual expression, such as the default value of a column. This helps to check the content of system catalogs. If the expression might contain keywords, specify the OID of the relationship they refer to as the second parameter; if no keywords are expected, zero is sufficient.

- pg_get_functiondef(func_oid)

  Description: Obtains the definition of a function.

  Return type: text

  Example:

```
gaussdb=# SELECT * FROM pg_get_functiondef(598);
 headerlines |                  definition
-------------+-----------------------------------------------------
           4 | CREATE OR REPLACE FUNCTION pg_catalog.abbrev(inet)+
             | RETURNS text                                      +
             | LANGUAGE internal                                 +
             | IMMUTABLE STRICT NOT FENCED NOT SHIPPABLE         +
             | AS $function$inet_abbrev$function$                +
             |
(1 row)
```

- pg_get_function_arguments(func_oid)

  Description: Obtains argument list of function's definition (with default values).

  Return type: text

  Note: **pg_get_function_arguments** returns the parameter list of a function, in the form it would need to appear in **CREATE FUNCTION**.

- pg_get_function_identity_arguments(func_oid)

  Description: Obtains the parameter list to identify a function (without default values).

  Return type: text

  Note: **pg_get_function_identity_arguments** returns the parameter list required to identify a function, in the form it would need to appear in **ALTER FUNCTION**. This form omits default values.

- pg_get_function_result(func_oid)

  Description: Obtains the RETURNS clause for a function.

  Return type: text

  Note: **pg_get_function_result** returns the appropriate **RETURNS** clause for the function.

- pg_get_indexdef(index_oid)

Description: Obtains the **CREATE INDEX** command for an index.

Return type: text

Example:

```
gaussdb=# SELECT * FROM pg_get_indexdef(16416);
                        pg_get_indexdef
-------------------------------------------------------------------------
 CREATE INDEX test3_b_idx ON test3 USING btree (b) TABLESPACE pg_default
(1 row)
```

- pg_get_indexdef(index_oid, dump_schema_only)

  Description: Obtains the **CREATE INDEX** command for indexes in dump scenarios. For an interval partitioned table that contains a local index, if **dump_schema_only** is set to **true**, the returned index creation statement does not contain the local index information of the automatically created partition. If **dump_schema_only** is set to **false**, the returned index creation statement contains the local index information of the automatically created partition. For a non-interval partitioned table or an interval partitioned table that does not contain a local index, the value of **dump_schema_only** does not affect the returned result of the function.

  Return type: text

  Example:

```
gaussdb=# CREATE TABLE sales
(prod_id NUMBER(6),
cust_id NUMBER,
time_id DATE,
channel_id CHAR(1),
promo_id NUMBER(6),
quantity_sold NUMBER(3),
amount_sold NUMBER(10,2))
PARTITION BY RANGE( time_id) INTERVAL('1 day')
(partition p1 VALUES LESS THAN ('2019-02-01 00:00:00'),
partition p2 VALUES LESS THAN ('2019-02-02 00:00:00')
);
CREATE TABLE
gaussdb=# CREATE INDEX index_sales ON sales(prod_id) local (PARTITION idx_p1 ,PARTITION idx_p2);
CREATE INDEX
gaussdb=# -- If the data to be inserted does not match any partition, create a partition and insert the
data into the new partition.
gaussdb=# INSERT INTO sales VALUES(1, 12, '2019-02-05 00:00:00', 'a', 1, 1, 1);
INSERT 0 1
gaussdb=# SELECT oid FROM pg_class WHERE relname = 'index_sales';
  oid
-------
 24632
(1 row)
gaussdb=# SELECT * FROM pg_get_indexdef(24632, true);
                                              pg_get_indexdef
-----------------------------------------------------------------------------------------------------------------------
---
 CREATE INDEX index_sales ON sales USING btree (prod_id) LOCAL(PARTITION idx_p1, PARTITION
idx_p2)  TABLESPACE pg_default
(1 row)
gaussdb=# SELECT * FROM pg_get_indexdef(24632, false);
                                                    pg_get_indexdef

-----------------------------------------------------------------------------------------------------------------------
-------------
--------------------
 CREATE INDEX index_sales ON sales USING btree (prod_id) LOCAL(PARTITION idx_p1, PARTITION
idx_p2, PARTITION sys_p1_prod_id_idx)  TA
BLESPACE pg_default
(1 row
```

- pg_get_indexdef(index_oid, column_no, pretty_bool)

Description: Obtains the **CREATE INDEX** command for an index, or definition of just one index column when the value of **column_no** is not zero.

Example:

```
gaussdb=# SELECT * FROM pg_get_indexdef(16416, 0, false);
                          pg_get_indexdef
---------------------------------------------------------------------------
 CREATE INDEX test3_b_idx ON test3 USING btree (b) TABLESPACE pg_default
(1 row)
gaussdb=# select * from pg_get_indexdef(16416, 1, false);
 pg_get_indexdef
-----------------
 b
(1 row)
```

Return type: text

Note: **pg_get_functiondef** returns a complete CREATE OR REPLACE FUNCTION statement for a function.

- pg_get_keywords()

  Description: Obtains the list of SQL keywords and their categories.

  Return type: SETOF record

  Note: **pg_get_keywords** returns a set of records describing the SQL keywords recognized by the server. The **word** column contains the keywords. The **catcode** column contains a category code: **U** for unreserved, **C** for column name, **T** for type or function name, or **R** for reserved. The **catdesc** column contains a possibly-localized string describing the category.

- pg_get_userbyid(role_oid)

  Description: Obtains the role name with a given OID.

  Return type: name

  Note: **pg_get_userbyid** extracts a role's name given its OID.

- pg_check_authid(role_id)

  Description: Checks whether a user exists based on **role_id**.

  Return type: text

  Example:

  ```
  gaussdb=# SELECT pg_check_authid(20);
  pg_check_authid
  -----------------
  f
  (1 row)
  ```

- pg_get_viewdef(view_name)

  Description: Obtains the underlying **SELECT** command for a view.

  Return type: text

- pg_get_viewdef(view_name, pretty_bool)

  Description: Obtains the underlying **SELECT** command for a view. Lines with columns are wrapped to 80 columns if **pretty_bool** is set to **true**.

  Return type: text

  Note: **pg_get_viewdef** reconstructs the SELECT query that defines a view. Most of these functions come in two forms. When the function has the **pretty_bool** parameter and the value is **true**, it can optionally "pretty-print" the result. The pretty-printed format is more readable. The other one is the default format which is more likely to be interpreted in the same way by future versions. Avoid using pretty-printed output for dump purposes. Passing

false to the pretty-print parameter generates the same result as a variant without this parameter.

- pg_get_viewdef(view_oid)

  Description: Obtains the underlying SELECT command for a view.

  Return type: text

- pg_get_viewdef(view_oid, pretty_bool)

  Description: Obtains the underlying **SELECT** command for a view. Lines with columns are wrapped to 80 columns if **pretty_bool** is set to **true**.

  Return type: text

- pg_get_viewdef(view_oid, wrap_column_int)

  Description: Obtains the underlying **SELECT** command for a view. Lines with columns are wrapped to the specified number of columns and printing is implicit.

  Return type: text

- pg_get_tabledef(table_oid)

  Description: Obtains the definition of a table based on **table_oid**.

  Example:

```
gaussdb=# SELECT * FROM pg_get_tabledef(16384);
                          pg_get_tabledef
-------------------------------------------------------------------------
 SET search_path = public;                                               +
 CREATE TABLE t1 (                                                       +
    c1 bigint DEFAULT nextval('serial'::regclass)                        +
 )                                                                       +
 WITH (orientation=row, compression=no, storage_type=USTORE, segment=off);+
 ILM Security Policies:                                                  +
   POLICY "p4" TABLE                                                     +
     ROW STORE COMPRESS ADVANCED ROW AFTER 1 DAYS OF NO MODIFICATION     +
     STATUS: ENABLED  DELETED: NO                                        +
(1 row)
```

  Return type: text

- pg_get_tabledef(table_name)

  Description: Obtains a table definition based on **table_name**.

  Example:

```
gaussdb=# SELECT * FROM pg_get_tabledef('t1');
                          pg_get_tabledef
-------------------------------------------------------------------------
 SET search_path = public;                                               +
 CREATE TABLE t1 (                                                       +
    c1 bigint DEFAULT nextval('serial'::regclass)                        +
 )                                                                       +
 WITH (orientation=row, compression=no, storage_type=USTORE, segment=off);+
 ILM Security Policies:                                                  +
   POLICY "p4" TABLE                                                     +
     ROW STORE COMPRESS ADVANCED ROW AFTER 1 DAYS OF NO MODIFICATION     +
     STATUS: ENABLED  DELETED: NO                                        +
(1 row)
```

  Return type: text

  Note: pg_get_tabledef reconstructs the CREATE statement of the table definition, including the table definition, index information, comments, and ILM policy (if exists). You need to separately create the dependent objects of the table, such as groups, schemas, tablespaces, and servers. The table definition does not include the statements for creating these dependent objects.

- pg_options_to_table(reloptions)

  Description: Obtains the set of storage option name/value pairs.

  Return type: SETOF record

  Note: **pg_options_to_table** returns the set of storage option name/value pairs (**option_name**/**option_value**) when **pg_class.reloptions** or **pg_attribute.attoptions** is passed.

- pg_tablespace_databases(tablespace_oid)

  Description: Obtains the set of database OIDs that have objects in the specified tablespace.

  Return type: setof oid

  Note: **pg_tablespace_databases** allows a tablespace to be checked. It returns the set of OIDs of databases that have objects stored in the tablespace. If this function returns any rows of data, the tablespace is not empty and cannot be dropped. To display the specific objects in the tablespace, you need to connect to the databases identified by **pg_tablespace_databases** and query the pg_class system catalogs.

- pg_tablespace_location(tablespace_oid)

  Description: Obtains the path in the file system that this tablespace is located in.

  Return type: text

- pg_typeof(any)

  Description: Obtains the data type of any value.

  Return type: regtype

  Note: **pg_typeof** returns the OID of the data type of the value that is passed to it. This can be helpful for troubleshooting or dynamically constructing SQL queries. It is declared that the return type of this function is regtype, which is an OID alias type (see **Object Identifier Types**). It is the same as an OID for comparison purposes but displays as a type name.

  Example:

```
gaussdb=# SELECT pg_typeof(33);
 pg_typeof
-----------
 integer
(1 row)

gaussdb=# SELECT typlen FROM pg_type WHERE oid = pg_typeof(33);
 typlen
--------
      4
(1 row)
```

- collation for (any)

  Description: Obtains the collation of the parameter.

  Return type: text

  Note: The expression **collation for** returns the collation of the value that is passed to it.

  The value might be quoted and schema-qualified. If no collation is derived for the parameter expression, then a null value is returned. If the argument is not of a collatable data type, then an error is raised.

  Example:


```
gaussdb=# SELECT collation for (description) FROM pg_description LIMIT 1;
 pg_collation_for
------------------
 "default"
(1 row)
```

- pg_extension_update_paths(name)

  Description: Returns the version update path of the specified extension.

  Return type: text(source text), text(path text), text(target text)

- pg_get_serial_sequence(tablename, colname)

  Description: Obtains the sequence of the corresponding table name and column name.

  Return type: text

  Example:

```
gaussdb=# select * from pg_get_serial_sequence('t1', 'c1');
 pg_get_serial_sequence
------------------------
 public.serial
(1 row)
```

- pg_sequence_parameters(sequence_oid)

  Description: Obtains the parameters of a specified sequence, including the start value, minimum value, maximum value, and incremental value.

  Return type: int16, int16, int16, int16, Boolean

  Example:

```
gaussdb=# select * from pg_sequence_parameters(16420);
 start_value | minimum_value |     maximum_value     | increment | cycle_option
-------------+---------------+-----------------------+-----------+--------------
         101 |             1 | 9223372036854775807 |         1 | f
(1 row)
```

Comment Information Functions

- col_description(table_oid, column_number)

  Description: Obtains the comment for a table column.

  Return type: text

  Note: **col_description** returns the comment for a table column, which is specified by the OID of its table and its column number.

- obj_description(object_oid, catalog_name)

  Description: Obtains the comment for a database object.

  Return type: text

  Note: The two-parameter form of **obj_description** returns the comment for a database object specified by its OID and the name of the system catalog to which it belongs. For example, **obj_description(123456,'pg_class')** would retrieve the comment for the table with OID 123456. The one-parameter form of **obj_description** requires only the OID.

  obj_description cannot be used for table columns since columns do not have OIDs of their own.

- obj_description(object_oid)

  Description: Obtains the comment for a database object.

  Return type: text

- shobj_description(object_oid, catalog_name)

  Description: Obtains the comment for a shared database object.

  Return type: text

  Note: **shobj_description** is used just like **obj_description**, except that the former is used for shared objects. Some system catalogs are global to all databases in the GaussDB, and the comments for objects in them are stored globally as well.

XIDs and Snapshots

Internal XIDs are 64 bits. **txid_snapshot**, data type used by these functions, stores information about XID visibility at a particular moment in time. **Table 7-90** describes the components.

Table 7-90 Snapshot components

Name	Description
xmin	Earliest XID (**txid**) that is still active. All earlier transactions will either be committed and visible, or rolled back.
xmax	First as-yet-unassigned **txid**. All **txids** greater than or equal to this are not yet started as of the time of the snapshot, so they are invisible.
xip_list	Active **txids** at the time of the snapshot. The list includes only those active **txids** between **xmin** and **xmax**; there might be active **txids** higher than **xmax**. A **txid** that is greater than or equal to **xmin** and less than **xmax** and that is not in this list was already completed at the time of the snapshot, and is either visible or rolled back according to its commit status. The list does not include **txids** of subtransactions.

The textual representation of **txid_snapshot** is **xmin:xmax:xip_list**.

For example, **10:20:10,14,15** means **xmin=10, xmax=20, xip_list=10, 14, 15**.

The following functions provide server transaction information in an exportable form. These functions are mainly used to determine which transactions were committed between two snapshots.

- pgxc_is_committed(transaction_id)

  Description: Specifies whether the given XID is committed or ignored. **NULL** indicates the unknown status (it can be running, ready, frozen, or other status).

  Return type: Boolean

- pgxc_is_committed(transaction_id, bucketid)

  Description: The hash bucket table is not supported in the current version in centralized mode. An error is reported when the API function is called.

- txid_current()

  Description: Obtains the current XID.

Return type: bigint

- gs_txid_oldestxmin()

  Description: Obtains the minimum XID (specified by **oldesxmin**).

  Return type: bigint

- txid_current_snapshot()

  Description: Obtains the current snapshot.

  Return type: txid_snapshot

- txid_snapshot_xip(txid_snapshot)

  Description: Obtains in-progress XIDs in a snapshot.

  Return type: setof bigint

- txid_snapshot_xmax(txid_snapshot)

  Description: Obtains **xmax** of snapshots.

  Return type: bigint

- txid_snapshot_xmin(txid_snapshot)

  Description: Obtains **xmin** of snapshots.

  Return type: bigint

- txid_visible_in_snapshot(bigint, txid_snapshot)

  Description: Specifies whether the XID is visible in a snapshot (excluding subtransaction IDs).

  Return type: Boolean

- get_local_prepared_xact()

  Description: Obtains the two-phase residual transaction information of the current node, including the XID, GID of the two-phase transaction, prepared time, owner OID, database OID, and node name of the current node.

  Return type: xid, text, timestamptz, oid, text

- get_remote_prepared_xacts()

  Description: Obtains the two-phase residual transaction information of all remote nodes, including the XID, GID of the two-phase transaction, prepared time, owner name, database name, and node name.

  Return type: xid, text, timestamptz, name, text

- global_clean_prepared_xacts(text, text)

  Description: Concurrently cleans two-phase residual transactions. Only the **gs_clean** tool can call this function for the cleaning in the distributed GaussDB environment. If other users call this function, **false** is returned.

  Return type: Boolean

- gs_get_next_xid_csn()

  Description: Returns the values of **next_xid** and **next_csn** on all nodes globally.

  Return value: See **Table 7-91**.

Table 7-91 gs_get_next_xid_csn parameter description

Column	Description
nodename	Node name.
next_xid	ID of the next transaction on the current node.
next_csn	Next CSN of the current node.

- pg_control_system()

  Description: Returns the status of the system control file.

  Return type: SETOF record
- pg_control_checkpoint()

  Description: Returns the system checkpoint status.

  Return type: SETOF record
- pv_builtin_functions

  Description: Displays information about all built-in system functions.

  Parameter: nan

  Return type: proname name, pronamespace oid, proowner oid, prolang oid, procost real, prorows real, provariadic oid, protransform regproc, proisagg boolean, proiswindow boolean, prosecdef boolean, proleakproof boolean, proisstrict boolean, proretset boolean, provolatile "char", pronargs smallint, pronargdefaults smallint, prorettype oid, proargtypes oidvector, proallargtypes integer[], proargmodes "char"[], proargnames text[], proargdefaults pg_node_tree, prosrc text, probin text, proconfig text[], proacl aclitem[], prodefaultargpos int2vector, fencedmode boolean, proshippable boolean, propackage boolean, oid oid
- pv_thread_memory_detail

  Description: Returns the memory information of each thread.

  Parameter: nan

  Return type: threadid text, tid bigint, thrdtype text, contextname text, level smallint, parent text, totalsize bigint, freesize bigint, usedsize bigint
- pg_relation_compression_ratio

  Description: Queries the compression rate of a table. By default, **1.0** is returned.

  Parameter: text

  Return type: real
- pg_relation_with_compression

  Description: Specifies whether a table is compressed.

  Parameter: text

  Return type: Boolean
- pg_stat_file_recursive

  Description: Lists all files in a path.

  Parameter: location text

- pg_shared_memory_detail

  Description: Returns usage information about all generated shared memory contexts. For details about each column, see **GS_SHARED_MEMORY_DETAIL**.

  Parameter: nan

  Return type: contextname text, level smallint, parent text, totalsize bigint, freesize bigint, usedsize bigint

- get_gtm_lite_status

  Description: Returns the backup XID and CSN on the GTM for fault locating. This system function is not supported in GTM-FREE mode or centralized mode.

- gs_stat_get_wlm_plan_operator_info

  Description: Obtains operator plan information from the internal hash table.

  Parameter: oid

  Return type: datname text, queryid int8, plan_node_id int4, startup_time int8, total_time int8, actual_rows int8, max_peak_memory int4, query_dop int4, parent_node_id int4, left_child_id int4, right_child_id int4, operation text, orientation text, strategy text, options text, condition text, projection text

- pg_stat_get_partition_tuples_hot_updated

  Description: Returns statistics on the number of hot updated tuples in a partition with a specified partition ID.

  Parameter: oid

  Return type: bigint

- gs_session_memory_detail_tp

  Description: Returns the memory usage of the session. For details, see **gs_session_memory_detail**.

  Parameter: nan

  Return type: sessid text, sesstype text, contextname text, level smallint, parent text, totalsize bigint, freesize bigint, usedsize bigint

- gs_thread_memory_detail

  Description: Returns the memory information of each thread.

  Parameter: nan

  Return type: threadid text, tid bigint, thrdtype text, contextname text, level smallint, parent text, totalsize bigint, freesize bigint, usedsize bigint

- pg_stat_get_wlm_session_iostat_info()

  Description: Returns the session load I/O information.

  Parameter: nan

  Return type: threadid bigint, maxcurr_iops integer, mincurr_iops integer, maxpeak_iops integer, minpeak_iops integer, iops_limits integer, io_priority integerw, and curr_io_limits integer

- adm_hist_snapshot_func()

  Description: Returns information about the snapshot execution time. To access this function, set the **enable_wdr_snapshot** parameter to **on** and obtain the permission to access the **snapshot schema**, **snapshot table**, and **tables_snap_timestamp** tables.

  Parameter: nan

Return type: snap_id bigint, dbid oid, begin_interval_time timestamp(3), end_interval_time timestamp(3), flush_elapsed interval day(5) to second(1), begin_interval_time_tz timestamp(3) with time zone, end_interval_time_tz timestamp(3) with time zone

- gs_get_current_version()

  Description: Returns the current compilation mode based on the current compilation macro. **'P'** is returned.

  Parameter: nan

  Return type: char

- gs_get_kernel_info()

  Description: Transaction information on DNs.

  Return value: See **Table 7-92**.

Table 7-92 gs_get_kernel_info parameter description

Name	Type	Description
node_name	text	Node name.
module	text	Module name, including: • **XACT** (transaction module) • **STANDBY** (standby module) • **UNDO** (undo module) • **HOTPATH** (hot patch module)

Name	Type	Description
name	text	Name of the key data in the memory state, including: • **startup_max_xid** (maximum XID when a thread is started) • **recent_local_xmin** (minimum XID of local active transactions) • **recent_global_xmin** (minimum XID of global active transactions) • **standby_xmin** (minimum XID of active transactions on the standby node) • **standby_redo_cleanup_xmin** (minimum XID of cleanup logs during redo on the standby node) • **standby_redo_cleanup_xmin_lsn** (LSN of the minimum XID of cleanup logs during redo on the standby node) • **local_csn_min** (minimum CSN of local active transactions) • **replication_slot_xmin** (minimum XID of replication slots) • **replication_slot_catalog_xmin** (minimum XID of catalog replication slots) • **global_recycle_xid** (minimum XID of global undo recycling transactions) • **global_frozen_xid** (minimum XID of global frozen transactions) • **recent_xmin** (minimum XID of active transactions in the current session) • **next_csn** (CSN of the next transaction) • **hotpatch_additional_info** (reserved column for hot patches) • **stmt_session_discard_records** (data volume discarded by full SQL statements due to full slots supported by the kernel) • **stmt_shm_flush_discard_records** (data volume discarded by full SQL statements due to full ringbuf supported by the kernel)
value	text	Value of the key data in the memory state.

Example:


```
gaussdb=# SELECT * FROM gs_get_kernel_info();
  node_name   | module |              name               |  value
--------------+--------+---------------------------------+---------
 datanode1    | XACT   | startup_max_xid                 | 16488
 datanode1    | XACT   | recent_local_xmin               | 15805
 datanode1    | XACT   | recent_global_xmin              | 15805
 datanode1    | STANDBY | standby_xmin                   | 0
 datanode1    | STANDBY | standby_redo_cleanup_xmin      | 0
 datanode1    | STANDBY | standby_redo_cleanup_xmin_lsn | 0/0
 datanode1    | XACT   | local_csn_min                   | 6014225
 datanode1    | XACT   | replication_slot_xmin           | 0
 datanode1    | XACT   | replication_slot_catalog_xmin | 0
 datanode1    | UNDO   | global_recycle_xid              | 15805
 datanode1    | XACT   | global_frozen_xid               | 0
 datanode1    | XACT   | recent_xmin                     | 15805
 datanode1    | XACT   | next_csn                        | 6014226
 datanode1    | HOTPATH | hotpatch_additional_info       |
 datanode1    | FULL_SQL| stmt_session_discard_records  | 0
 datanode1    | FULL_SQL| stmt_shm_flush_discard_records| 0
(16 row)
```

7.6.26 System Administration Functions

7.6.26.1 Configuration Settings Functions

Configuration setting functions are used for querying and modifying GUC parameters.

- current_setting(setting_name)

  Description: Specifies the current setting.

  Return type: text

  Note: **current_setting** obtains the current setting of **setting_name** by query. It is equivalent to the **SHOW** statement. For example:

  ```
  gaussdb=# SELECT current_setting('datestyle');

   current_setting
  -----------------
   ISO, MDY
  (1 row)
  ```

- set_working_grand_version_num_manually(tmp_version)

  Description: Upgrades new features of the database by switching the authorization version.

  Return type: void

- shell_in(type)

  Description: Inputs a route for the shell type that has not yet been filled.

  Return type: void

- shell_out(type)

  Description: Outputs a route for the shell type that has not yet been filled.

  Return type: void

- set_config(setting_name, new_value, is_local)

  Description: Sets the parameter and returns a new value.

  Return type: text

  Note: The **set_config** parameter sets the value of **setting_name** to **new_value**. If **is_local** is **true**, the new value applies only to the current

transaction. If you want the new value to apply to the current session, use **false** instead. This function is equivalent to the SET statement. Example:

```
gaussdb=# SELECT set_config('log_statement_stats', 'off', false);

 set_config
------------
 off
(1 row)
```

7.6.26.2 Universal File Access Functions

Universal file access functions provide local access APIs for files on a database server. Only files in the database directory and the **log_directory** directory can be accessed. A relative path is used for files in the database directory, and a path matching the **log_directory** configuration setting is used for log files. Only database initialization users can use these functions.

- pg_ls_dir(dirname text)

  Description: Lists files in a directory.

  Return type: setof text

  Note: pg_ls_dir returns all the names in the specified directory, except the special entries "." and "..".

  Example:

```
gaussdb=# SELECT pg_ls_dir('./');
      pg_ls_dir
----------------------
 .gaussdb.conf.swp
 gaussdb.conf
 pg_tblspc
 PG_VERSION
 gs_ident.conf
 core
 server.crt
 pg_serial
 pg_twophase
 gaussdb.conf.lock
 pg_stat_tmp
 pg_notify
 pg_subtrans
 pg_ctl.lock
 pg_xlog
 pg_clog
 base
 pg_snapshots
 postmaster.opts
 postmaster.pid
 server.key.rand
 server.key.cipher
 pg_multixact
 pg_errorinfo
 server.key
 gs_hba.conf
 pg_replslot
 .gs_hba.conf.swp
 cacert.pem
 gs_hba.conf.lock
 global
 gaussdb.state
(32 rows)
```

- pg_read_file(filename text, offset bigint, length bigint)

  Description: Returns the content of a text file.

Return type: text

Note: pg_read_file returns part of a text file. It can return a maximum of *length* bytes from *offset*. The actual size of fetched data is less than *length* if the end of the file is reached first. If *offset* is negative, it is the length rolled back from the file end. If *offset* and *length* are omitted, the entire file is returned.

Example:

```
gaussdb=# SELECT pg_read_file('postmaster.pid',0,100);
           pg_read_file
----------------------------------------
 53078                                 +
 /srv/BigData/hadoop/data1/dbnode+
 1500022474                            +
 8000                                 +
 /var/run/FusionInsight               +
 localhost                            +
  2
(1 row)
```

- pg_read_binary_file(filename text [, offset bigint, length bigint,missing_ok boolean])

  Description: Returns the content of a binary file.

  Return type: bytea

  Note: pg_read_binary_file is similar to pg_read_file, except that the result is a bytea value; accordingly, no encoding checks are performed. In combination with the convert_from function, this function can be used to read a file in a specified encoding.

  ```
  gaussdb=# SELECT convert_from(pg_read_binary_file('filename'), 'UTF8');
  ```

- pg_read_binary_file_blocks(filename text, blocknum bigint, blockcount bigint)

  Description: Returns the binary content of a specified page area of a compressed file.

  Return type: bytea

  Note: pg_read_binary_file_blocks is used only to read the binary content of a compressed file.

- pg_stat_file(filename text)

  Description: Returns status information about a file.

  Return type: record

  Note: pg_stat_file returns a record containing the file size, last access timestamp, last modification timestamp, last file status change timestamp, and a Boolean value indicating if it is a directory. Typical usages are as follows:

  ```
  gaussdb=# SELECT * FROM pg_stat_file('filename');
  gaussdb=# SELECT (pg_stat_file('filename')).modification;
  ```

  Example:

  ```
  gaussdb=# SELECT convert_from(pg_read_binary_file('postmaster.pid'), 'UTF8');
             convert_from
  ----------------------------------------
   4881                                 +
   /srv/BigData/gaussdb/data1/dbnode+
   1496308688                           +
   25108                                +
   /opt/user/Bigdata/gaussdb/gaussdb_tmp +
   *                                   +
    25108001  43352069                  +
  ```



```
(1 row)
gaussdb=# SELECT * FROM pg_stat_file('postmaster.pid');

 size |         access         |      modification      |         change
| creation | isdir
------+------------------------+------------------------+------------------------
+----------+-------
  117 | 2017-06-05 11:06:34+08 | 2017-06-01 17:18:08+08 | 2017-06-01 17:18:08+08
|          | f
(1 row)
gaussdb=# SELECT (pg_stat_file('postmaster.pid')).modification;
      modification
------------------------
 2017-06-01 17:18:08+08
(1 row)
```

7.6.26.3 Server Signal Functions

Server signal functions send control signals to other server threads. Only a system administrator has the permission to execute the following functions:

- pg_cancel_backend(pid int)

  Description: Cancels a statement that is being executed by a backend thread.

  Return type: Boolean

  Note: pg_cancel_backend sends a query cancellation (SIGINT) signal to the backend thread identified by **pid**. The PID of an active backend thread can be found in the **pid** column of the pg_stat_activity view, or can be found by listing the database thread using **ps** on the server. A user with the SYSADMIN permission, the owner of the database connected to the backend thread, the owner of the backend thread, or a user who inherits the built-in role permission gs_role_signal_backend can use this function.

- pg_cancel_session(pid bigint, sessionid bigint)

  Description: Cancels the statement that is being executed by an active session in thread pool mode.

  Return type: Boolean

  Note: The input parameters of pg_cancel_session can be queried using the **pid** and **sessionid** columns in pg_stat_activity. The input parameters can be used to cancel the statements that are being executed by active sessions in thread pool mode. When the input parameters **pid** and **sessionid** are the same and both are thread IDs, the function of pg_cancel_backend is the same as that of pg_cancel_backend.

- pg_reload_conf()

  Description: Causes all server threads to overload their configuration files.

  Return type: Boolean

  Note: pg_reload_conf sends a SIGHUP signal to the server. As a result, all server threads overload their configuration files.

- pg_rotate_logfile()

  Description: Rotates the log files of the server.

  Return type: Boolean

  Note: pg_rotate_logfile sends a signal to the log file manager, instructing the manager to immediately switch to a new output file. This function works only

when redirect_stderr is used for log output. Otherwise, no log file manager sub-thread is generated.

- pg_terminate_backend(pid int)

  Description: Terminates a backend thread.

  Return type: Boolean

  Note: Each of these functions returns **true** if they are successful and **false** otherwise. A user with the SYSADMIN permission, the owner of the database connected to the backend thread, the owner of the backend thread, or a user who inherits the built-in role permission gs_role_signal_backend can use this function.

  NOTICE

  This function can terminate non-thread pool threads and active thread pool threads, but cannot terminate inactive thread pool threads.

  Example:

```
gaussdb=# SELECT pid from pg_stat_activity;
       pid
-----------------
 140657876268816
(1 rows)

gaussdb=# SELECT pg_terminate_backend(140657876268816);
 pg_terminate_backend
----------------------
 t
(1 row)
```

- pg_terminate_session(pid int64, sessionid int64)

  Description: Terminates a backend session in thread pool mode.

  Return type: Boolean

  Note: Each of these functions returns **true** if they are successful and **false** otherwise. A user with the SYSADMIN permission, the owner of the database connected to the session, the owner of the session, or a user who inherits the built-in role permission gs_role_signal_backend has the permission to use this function.

  NOTICE

  When the input parameters **pid** and **sessionid** are the same and both are thread IDs, this function can terminate non-thread pool threads and active thread pool threads.

  When the input parameters **pid** and **sessionid** are different, this function can terminate active sessions or close inactive sessions and the socket connection of the client.

- pg_terminate_active_session_socket(pid int64, sessionid int64)

  Description: Closes the socket connection between an active session and the client.

  Return type: Boolean

Note: Each of these functions returns **true** if they are successful and **false** otherwise. This function can be used only by initial users.

7.6.26.4 Backup and Restoration Control Functions

Backup Control Functions

Backup control functions help with online backup.

- pg_create_restore_point(name text)

  Description: Creates a named point for performing a restore (requires an administrator role).

  Return type: text

  Note: pg_create_restore_point creates a named transaction log record that can be used as a restoration target, and returns the corresponding transaction log location. The given name can then be used with **recovery_target_name** to specify the point up to which restoration will proceed. Avoid creating multiple restoration points with the same name, since restoration will stop at the first one whose name matches the restoration target.

- pg_current_xlog_location()

  Description: Obtains the write position of the current transaction log.

  Return type: text

  Note: pg_current_xlog_location displays the write position of the current transaction log in the same format as those of the previous functions. Read-only operations do not require permissions of the system administrator.

- pg_current_xlog_insert_location()

  Description: Obtains the insert position of the current transaction log.

  Return type: text

  Note: **pg_current_xlog_insert_location** displays the insert position of the current transaction log. The insertion point is the logical end of the transaction log at any instant, while the write location is the end of what has been written out from the server's internal buffers. The write position is the end that can be detected externally from the server. This operation can be performed to archive only some of completed transaction log files. The insert position is mainly used for commissioning the server. Read-only operations do not require permissions of the system administrator.

- gs_current_xlog_insert_end_location()

  Description: Obtains the insert position of the current transaction log.

  Return type: text

  Note: **gs_current_xlog_insert_end_location** displays the insert position of the current transaction log.

- pg_start_backup(label text [, fast boolean])

  Description: Starts to perform online backup. (An administrator, replication role, or O&M administrator must enable **operation_mode**.)

  Return type: text

  Note: pg_start_backup receives a user-defined backup label (usually the name of the position where the backup dump file is stored). This function writes a

backup label file to the data directory of the database and then returns the starting position of backed up transaction logs in text mode. This function must be used together with pg_stop_backup(). If this function is called independently, backup_label remains. During WAL replay, the checkpoint is read based on backup_label. If the WAL file corresponding to the checkpoint has been reclaimed, the database cannot be started.

```
gaussdb=# SELECT pg_start_backup('label_goes_here');
 pg_start_backup
-----------------
 0/3000020
(1 row)
```

- pg_stop_backup()

  Description: Completes the online backup. (An administrator, replication role, or O&M administrator must enable **operation_mode**.)

  Return type: text

  Note: **pg_stop_backup** deletes the label file created by **pg_start_backup** and creates a backup history file in the transaction log archive area. The history file includes the label given to **pg_start_backup**, the start and end transaction log locations for the backup, and the start and end time of the backup. The return value is the backup's ending transaction log location. After the end position is calculated, the insert position of the current transaction log automatically goes ahead to the next transaction log file. The ended transaction log file can be immediately archived so that backup is complete.

- pg_switch_xlog()

  Description: Switches to a new transaction log file. (An administrator or O&M administrator must enable **operation_mode**.)

  Return type: text

  Note: pg_switch_xlog moves to the next transaction log file so that the current log file can be archived (if continuous archive is used). The return value is the ending transaction log location + 1 within the just-completed transaction log file. If there has been no transaction log activity since the last transaction log switchover, pg_switch_xlog does not move but returns the start location of the transaction log file currently in use.

- pg_xlogfile_name(location text)

  Description: Converts the position string in a transaction log to a file name.

  Return type: text

  Note: **pg_xlogfile_name** extracts only the transaction log file name. If the given transaction log position is the transaction log file border, a transaction log file name will be returned for both the two functions. This is usually the desired behavior for managing transaction log archiving, since the preceding file is the last one that currently needs to be archived.

- pg_xlogfile_name_offset(location text)

  Description: Converts the position string in a transaction log to a file name and returns the byte offset in the file.

  Return type: text, integer

  Note: **pg_xlogfile_name_offset** can extract transaction log file names and byte offsets from the returned results of the preceding functions. Example:

```
gaussdb=# SELECT * FROM pg_xlogfile_name_offset(pg_stop_backup());
NOTICE:  pg_stop_backup cleanup done, waiting for required WAL segments to be archived
```



```
NOTICE:  pg_stop_backup complete, all required WAL segments have been archived
        file_name         | file_offset
--------------------------+-------------
000000010000000000000003 |         272
(1 row)
```

- pg_xlog_location_diff(location text, location text)

  Description: Calculates the difference in bytes between two transaction log locations.

  Return type: numeric

- pg_cbm_start_tracked_location()

  Description: Queries the start LSN parsed by CBM.

  Return type: text

- pg_cbm_tracked_location()

  Description: Queries the LSN location parsed by CBM.

  Return type: text

- pg_cbm_get_merged_file(startLSNArg text, endLSNArg text)

  Description: Combines CBM files within the specified LSN range into one and returns the name of the combined file.

  Return type: text

  Note: Only the system administrator or O&M administrator can obtain the CBM combination file.

- pg_cbm_get_changed_block(startLSNArg text, endLSNArg text)

  Description: Combines CBM files within the specified LSN range into a table and return records of this table.

  Return type: record

  Note: The table columns returned by pg_cbm_get_changed_block include the start LSN, end LSN, tablespace OID, database OID, table relfilenode, table fork number, whether the table is a system catalog, whether the table is deleted, whether the table is created, whether the table is truncated, number of pages in the truncated table, number of modified pages, and list of modified page numbers.

- pg_cbm_recycle_file(targetLSNArg text)

  Description: Deletes the CBM files that are no longer used and returns the first LSN after the deletion.

  Return type: text

- pg_cbm_force_track(targetLSNArg text,timeOut int)

  Description: Forcibly executes the CBM trace to the specified Xlog position and returns the Xlog position of the actual trace end point.

  Return type: text

- pg_enable_delay_ddl_recycle()

  Description: Enables DDL delay and returns the Xlog position of the enabling point. An administrator or O&M administrator must enable **operation_mode**.

  Return type: text

- pg_disable_delay_ddl_recycle(barrierLSNArg text, isForce bool)

Description: Disables DDL delay and returns the Xlog range where DDL delay takes effect. An administrator or O&M administrator must enable **operation_mode**.

Return type: record

- pg_enable_delay_xlog_recycle()

  Description: Enables Xlog recycle delay. This function is used in primary database node restoration. An administrator or O&M administrator must enable **operation_mode**.

  Return type: void

- pg_disable_delay_xlog_recycle()

  Description: Disables Xlog recycle delay. This function is used in primary database node restoration. An administrator or O&M administrator must enable **operation_mode**.

  Return type: void

- pg_cbm_rotate_file(rotate_lsn text)

  Description: Forcibly switches the file after the CBM parses **rotate_lsn**. This function is called during the build process.

  Return type: void

- gs_roach_stop_backup(backupid text)

  Description: Stops a backup started by the internal backup tool GaussRoach. It is similar to the pg_stop_backup system function but is more lightweight.

  Return type: text. The content is the insertion position of the current log.

- gs_roach_enable_delay_ddl_recycle(backupid name)

  Description: Enables DDL delay and returns the log location of the enabling point. It is similar to the pg_enable_delay_ddl_recycle system function but is more lightweight. In addition, different **backupid** values can be used to concurrently open DDL statements with delay.

  Return type: text. The content is the log location of the start point.

- gs_roach_disable_delay_ddl_recycle(backupid text)

  Description: Disables DDL delay, returns the range of logs on which DDL delay takes effect. It is similar to the pg_enable_delay_ddl_recycle system function but is more lightweight. In addition, the DDL delay function can be disabled concurrently by specifying different backupid values.

  Return type: record. The content is the range of logs for which DDL is delayed to take effect.

- gs_roach_switch_xlog(request_ckpt bool)

  Description: Switches the currently used log segment file and triggers a full checkpoint if **request_ckpt** is set to **true**.

  Return type: text. The content is the location of the segment log.

- gs_block_dw_io(timeout int, identifier text)

  Description: Blocks doublewrite page flushing.

  Parameter description:

  – timeout

    Block duration.

Value range: [0,3600], in seconds. The value **0** indicates that the block duration is 0s.

– identifier

ID of the operation.

Value range: a string, supporting only uppercase letters, lowercase letters, digits, and underscores (_).

Return type: Boolean

Note: To call this function, the user must have the SYSADMIN or OPRADMIN permission. An O&M administrator must enable **operation_mode**.

- gs_is_dw_io_blocked()

  Description: Checks whether disk flushing on the current doublewrite page is blocked. If disk flushing is blocked, **true** is returned.

  Return type: Boolean

  Note: To call this function, the user must have the SYSADMIN or OPRADMIN permission. An O&M administrator must enable **operation_mode**.

- gs_pitr_advance_last_updated_barrier()

  Description: In PITR mode, forcibly pushes the global maximum archived recovery point uploaded to OBS/NAS last time to the current point. No input parameter is required.

  Return type: text

  Note: To call this function, the user must have the SYSADMIN or OPRADMIN permission. An O&M administrator must enable **operation_mode**. This parameter is valid only on the primary DN in a centralized system. The return value is the latest local maximum archived recovery point.

- gs_pitr_clean_local_barrier_files('delete_timestamp')

  Description: Clears locally cached barrier record files.

  Value range: The **delete_timestamp** parameter is of the text type. It is a Linux timestamp and contains 10 characters.

  Return type: text

  Note: To call this function, the user must have the SYSADMIN or OPRADMIN permission. An O&M administrator must enable **operation_mode**. The returned result is the start timestamp of the earliest barrier file on the local host after the deletion.

- gs_get_barrier_lsn(barrier_name text)

  Description: Obtains the LSN corresponding to the barrier created using a backup.

  Return type: text

  Note: Currently, this function is not supported. Currently, only **gs_roach_full** and **gs_roach_inc** are supported. To call this function, the user must have the SYSADMIN or OPRADMIN permission. An O&M administrator must enable **operation_mode**.

- gs_gbr_relation_associated_filenode(schemaName name, tableName name)

  Description: Returns the relfilenode of all indexes, sequences, partitions, TOAST tables, and TOAST indexes related to the input table.

  Return type: record

Note: The table columns returned by gs_gbr_relation_associated_filenode include the file type relkind, namespace where the file is located, relation name corresponding to the file, OID of the database where the file is located, OID of the tablespace where the file is located, and relfilenode of the file.

- pg_create_physical_replication_slot_extern(slotname text, dummy_standby bool, extra_content text, need_recycle_xlog bool)

  Description: Creates an OBS or a NAS archive slot. **slotname** indicates the name of the archive slot or recovery slot. The primary and standby slots must use the same slot name. **dummy_standby** is a reserved parameter. **extra_content** contains some information about the archive slot. For an OBS archive slot, the format is **OBS;obs_server_ip;obs_bucket_name;obs_ak;obs_sk;archive_path;is_recovery;is_vote_replicate**, in which **OBS** indicates the archive media of the archive slot, **obs_server_ip** indicates the IP address of OBS, **obs_bucket_name** indicates the bucket name, **obs_ak** indicates the AK of OBS, **obs_sk** indicates the SK of OBS, **archive_path** indicates the archive path, and **is_recovery** specifies whether the slot is an archive slot or a recovery slot (**0**: archive slot; **1**: recovery slot). **is_vote_replicate** indicates whether the voting copy is archived first. The value **0** indicates that the synchronous standby server is archived first, and the value **1** indicates that the voting copy is archived first. This field is reserved in the current version and is not adapted yet. For a NAS archive slot, the format is **NAS;archive_path;is_recovery;is_vote_replicate**. Compared with the OBS archive slot, the NAS archive slot does not have the OBS configuration information, while the meanings of other fields are the same.

  If the media is not OBS or NAS, the OBS archive slot is used by default. The **extra_content** format is **obs_server_ip;obs_bucket_name;obs_ak;obs_sk;archive_path;is_recovery;is_vote_replicate**.

  need_recycle_xlog specifies whether to recycle old archived logs when creating an archive slot. The value **true** indicates that old archived logs are recycled, and the value **false** indicates that old archive logs are not recycled.

  Return type: record contains slotname and xlog_position

  Note: Users who call this function must have the SYSADMIN permission or the REPLICATION permission, or inherit the gs_role_replication permission of the built-in role. Currently, multiple archive slots cannot be created.

  Examples:

  Create an OBS archive slot.

  ```
  gaussdb=# SELECT * FROM pg_create_physical_replication_slot_extern('uuid', false, 'OBS;obs.cn-north-7.ulanqab.huawei.com;dyk;19D772JBCACXX3KWS51D;********;gaussdb_uuid/dn1;0;0', false);
   slotname | xlog_position
  ----------+---------------
   uuid     |
  (1 row)
  ```

  Create a NAS archive slot.

  ```
  gaussdb=# SELECT * FROM pg_create_physical_replication_slot_extern('uuid', false, 'NAS;/data/nas/media/gaussdb_uuid/dn1;0;0',false);
   slotname | xlog_position
  ----------+---------------
   uuid     |
  ```

- gs_set_obs_delete_location(delete_location text)

Description: Sets the location where OBS archive logs can be deleted. The value of **delete_location** is an LSN. The logs before this location have been replayed and flushed to disks and can be deleted on OBS.

Return type: xlog_file_name text, indicating the file name of the logs that can be deleted. The value of this parameter is returned regardless of whether OBS is deleted successfully.

```
gaussdb=# SELECT gs_set_obs_delete_location('0/54000000');
 gs_set_obs_delete_location
----------------------------
 000000010000000000000054_00
(1 row)
```

- gs_set_obs_delete_location_with_slotname(cstring, cstring)

  Description: Sets the location where OBS archive logs in a specified archive slot can be deleted. The first parameter indicates the LSN. The logs before this location have been replayed and flushed to disks and can be deleted on OBS. The second parameter indicates the name of the archive slot.

  Return type: xlog_file_name text, indicating the file name of the logs that can be deleted. The value of this parameter is returned regardless of whether OBS is deleted successfully.

- gs_get_global_barrier_status()

  Description: gs_get_global_barrier_status is used to query the latest global barrier archived in OBS.

  Return type: text

  global_barrier_id: globally latest barrier ID.

  global_achive_barrier_id: globally latest archived barrier ID.

- gs_get_global_barriers_status()

  Description: gs_get_global_barriers_status is used to query the latest global barrier archived in OBS.

  Return type: text

  slot_name: slot name.

  global_barrier_id: globally latest barrier ID.

  global_achive_barrier_id: globally latest archived barrier ID.

Restoration Control Functions

Restoration control functions provide information about the status of standby nodes. These functions may be executed both during restoration and in normal running.

- pg_is_in_recovery()

  Description: Returns **true** if restoration is still in progress.

  Return type: Boolean

- pg_last_xlog_receive_location()

  Description: Obtains the last transaction log location received and synchronized to disk by streaming replication. While streaming replication is in progress, this will increase monotonically. If restoration has been completed, then this value will remain static at the value of the last WAL record received and synchronized to disk during restoration. If streaming

replication is disabled or if it has not yet started, the function returns a null value.

Return type: text

- pg_last_xlog_replay_location()

  Description: Obtains last transaction log location replayed during restoration. If restoration is still in progress, this will increase monotonically. If restoration has been completed, then this value will remain static at the value of the last WAL record received during that restoration. When the server has been started normally without restoration, the function returns a null value.

  Return type: text

- pg_last_xact_replay_timestamp()

  Description: Obtains the timestamp of last transaction replayed during restoration. This is the time to commit a transaction or abort a WAL record on the primary node. If no transactions have been replayed during restoration, this function will return a null value. If restoration is still in progress, this will increase monotonically. If restoration has been completed, then this value will remain static at the value of the last WAL record received during that restoration. If the server normally starts without manual intervention, this function will return a null value.

  Return type: timestamp with time zone

Restoration control functions control restoration threads. These functions may be executed only during restoration.

- pg_is_xlog_replay_paused()

  Description: Returns **true** if restoration is paused.

  Return type: Boolean

- pg_xlog_replay_pause()

  Description: Pauses restoration immediately.

  Return type: void

- pg_xlog_replay_resume()

  Description: Restarts restoration if it was paused.

  Return type: void

- gs_get_active_archiving_standby()

  Description: Queries information about archive standby nodes in the same shard. The standby node name, archive location, and number of archived logs are returned.

  Return type: text, int

- gs_pitr_get_warning_for_xlog_force_recycle()

  Description: Checks whether logs are recycled because a large number of logs are stacked in the archive slot after archiving is enabled.

  Return type: Boolean

- gs_pitr_clean_history_global_barriers(stop_barrier_timestamp cstring)

  Description: Clears all barrier records generated before the specified time. The earliest barrier record is returned. The input parameter is of the cstring type and is a Linux timestamp. You need to perform this operation as an administrator or O&M administrator.

Return type: text

- gs_pitr_archive_slot_force_advance(stop_barrier_timestamp cstring)

  Description: Forcibly pushes the archive slot and clears unnecessary barrier records. The new archive slot location is returned. The input parameter is of the cstring type and is a Linux timestamp. You need to perform this operation as an administrator or O&M administrator.

  Return type: text

- gs_recent_barrier_buffer_info(start_time text, end_time text)

  Description: Queries barrier information based on the time range entered by the user to obtain time_stamp, CSN, LSN, and standard_time.

  Return type: record

  Note: To call this function, you must have the SYSADMIN or OPRADMIN permission. The input parameters **start_time** and **end_time** are in the format of year-month-day time, where the time is in the clock format. The maximum query time span is one day. If the time span exceeds the limit, the end time is automatically converted to the limit boundary based on the query start time. Example:

```
gaussdb=# SELECT * FROM gs_recent_barrier_buffer_info('2024-01-15 23:27:50', '2024-01-15
23:28:00');
 timestamp  |       lsn        |   csn    |    standard_time
------------+------------------+----------+---------------------
 1705332470 | 00000000/15FFBBA0 | 41020421 | 2024-01-15 23:27:50
 1705332471 | 00000000/15FFBDF0 | 41020422 | 2024-01-15 23:27:51
 1705332472 | 00000000/15FFC058 | 41020423 | 2024-01-15 23:27:52
 1705332472 | 00000000/15FFC0F8 | 41020424 | 2024-01-15 23:27:52
 1705332473 | 00000000/15FFC348 | 41020425 | 2024-01-15 23:27:53
 1705332474 | 00000000/15FFC598 | 41020426 | 2024-01-15 23:27:54
 1705332475 | 00000000/15FFC638 | 41020427 | 2024-01-15 23:27:55
 1705332476 | 00000000/15FFC888 | 41020428 | 2024-01-15 23:27:56
 1705332476 | 00000000/15FFDC80 | 41020433 | 2024-01-15 23:27:56
 1705332477 | 00000000/15FFDD20 | 41020434 | 2024-01-15 23:27:57
 1705332478 | 00000000/15FFDF70 | 41020435 | 2024-01-15 23:27:58
 1705332479 | 00000000/15FFE1D8 | 41020436 | 2024-01-15 23:27:59
 1705332480 | 00000000/15FFE278 | 41020437 | 2024-01-15 23:28:00
 1705332480 | 00000000/15FFE4C8 | 41020438 | 2024-01-15 23:28:00
(14 rows)
```

- gs_show_obs_media_files(slot_name cstring, src cstring, offset int32, limit int32)

  Description: Queries the OBS file list based on the archive slot (**slot_name**) and OBS directory address (**src**) entered by the user.

  Return type: record

  Note: To call this function, you must have the SYSADMIN or OPRADMIN permission. **offset** indicates the query result offset, and **limit** indicates the maximum number of output lines. All files in **src** are queried. Example:

```
gaussdb=# SELECT gs_show_obs_archive_files('ssh','dn1/pg_xlog',0, 5);
                                gs_show_obs_archive_files
-----------------------------------------------------------------------------------------------
 (wstdist_ssh/archive/dn1/pg_xlog/
000000010000000000000007_00_01_00000004_00000002_00000000)
 (wstdist_ssh/archive/dn1/pg_xlog/
000000010000000000000007_00_01_00000103_00000003_00000000)
 (wstdist_ssh/archive/dn1/pg_xlog/
000000010000000000000007_01_01_00000004_00000002_00000000)
 (wstdist_ssh/archive/dn1/pg_xlog/
000000010000000000000007_01_01_00000103_00000003_00000000)
 (wstdist_ssh/archive/dn1/pg_xlog/
000000010000000000000007_02_01_00000004_00000002_00000000)
(5 rows)
```


- gs_upload_obs_media_file(slot_name cstring, src cstring, dest cstring, is_forced bool)

  Description: Uploads OBS files based on the archive slot (**slot_name**), source address (**src**), OBS address (**dest**), and whether to forcibly upload files (**is_forced**).

  Return type: void

  Note: To call this function, you must have the SYSADMIN or OPRADMIN permission. The original file directory must be the directory specified by *$GAUSSLOG*. Example:

```
gaussdb=# SELECT * FROM gs_upload_obs_archive_file('ssh', '/data/gauss/log/stwang/test/
000000010000000000000007_02_01_00000004_00000002_00000000', 'dn1/pg_xlog/
000000010000000000000019_02_01_00000000_00000000_00000003', true);
 gs_upload_obs_archive_file
----------------------------

(1 row)
```

- gs_download_obs_media_file(slot_name cstring, src cstring, dest cstring)

  Description: Downloads OBS files based on the archive slot (**slot_name**), download source address (**src**), and local destination address (**dest**).

  Return type: void

  Note: To call this function, you must have the SYSADMIN or OPRADMIN permission. The download directory must be the directory specified by *$GAUSSLOG*. Example:

```
gaussdb=# SELECT * FROM gs_download_obs_archive_file('ssh','dn1/pg_xlog/
000000010000000000000019_02_01_00000000_00000000_00000003','/data/gauss/log/stwang/test');
 gs_download_obs_archive_file
------------------------------

(1 row)
```

While restoration is paused, no further database changes are applied. In hot standby mode, all new queries will see the same consistent snapshot of the database, and no further query conflicts will be generated until restoration is resumed.

If streaming replication is disabled, the paused state may continue indefinitely without problem. While streaming replication is in progress, WAL records will continue to be received, which will eventually fill available disk space. This progress depends on the duration of the pause, the rate of WAL generation, and available disk space.

7.6.26.5 DR Control Functions for Dual-Database Instances

- gs_streaming_dr_in_switchover()

  Description: Truncates services during a planned switchover in the primary database instance in streaming replication-based remote DR solutions.

  Return type: Boolean, indicating whether the service truncation is successful and whether the switchover process can be performed normally.

7.6.26.6 DR Query Functions for Dual-Database Instances

- gs_get_local_barrier_status()

  Description: If two-city 3DC DR is enabled, logs of the primary database instance and DR database instance are synchronized. The progress of the

primary database instance's archived logs and the standby database instance's log replay is determined by flushing barrier logs to the primary database instance and replaying them on the standby database instance. **gs_get_local_barrier_status** is used to query the current log replay status of each node of the DR database instance.

Return type: text

barrier_id: latest barrier ID of a node of the DR database instance.

barrier_lsn: LSN of the latest barrier ID returned by a node in the DR database instance.

archive_lsn: location of archived logs obtained by a node in the DR database instance. This parameter does not take effect currently.

flush_lsn: location of logs that have been flushed to disks on a node in the DR database instance.

- gs_hadr_in_recovery()

  Description: Checks whether the current node is in barrier-based log restoration if two-city 3DC DR is enabled. If the current node is in barrier-based log restoration, **true** is returned. The DR database instance can be promoted to the production database instance through the switchover process only after the log restoration is complete (restricted to the system administrator).

  Return type: Boolean

  Note: This function is used only when a planned switchover is performed in the DR database instance.

- gs_streaming_dr_get_switchover_barrier()

  Description: Checks whether the DN instance of the DR database instance has received the switchover barrier logs and replayed the logs in two-city 3DC DR solutions based on streaming replication. If it has, **true** is returned. In the DR database instance, the procedure for promoting the DR database instance to the production database instance in the switchover process can be started only after the switchover barrier logs of all DN instances are replayed (restricted to the system administrator).

  Return type: Boolean

  Note: This function is used only when a planned switchover is performed in the DR database instance in streaming DR solutions.

- gs_streaming_dr_service_truncation_check()

  Description: Checks whether the DN instance of the primary database instance has sent the switchover barrier logs in two-city 3DC DR solutions based on streaming replication. If it has, **true** is returned. The procedure for demoting the production database instance to the DR database instance in the switchover process can be started only after the logs are sent (restricted to the system administrator).

  Return type: Boolean

  Note: This function is used only when a planned switchover is performed in the DR database instance.

- gs_hadr_local_rto_and_rpo_stat()

  Description: Displays the log flow control information about the primary and standby database instances in the streaming DR scenario. (This view can be

used only on the primary DN of the primary database instance. Statistics cannot be obtained from the standby DN or the standby database instance.)

The return value type is record. The types and meanings of the columns are as described in **Table 7-93**.

Table 7-93 Parameters of gs_hadr_local_rto_and_rpo_stat

Parameter	Type	Description
hadr_sender_node_name	text	Node name, including the primary database instance and the first standby node of the standby database instance.
hadr_receiver_node_name	text	Name of the first standby node of the standby database instance.
source_ip	text	IP address of the primary DN of the primary database instance.
source_port	int	Communication port of the primary DN of the primary database instance.
dest_ip	text	IP address of the first standby DN of the standby database instance.
dest_port	int	Communication port of the first standby DN of the standby database instance.
current_rto	int	Flow control information, that is, log RTO time of the current primary and standby database instances (unit: second).
target_rto	int	Flow control information, that is, the RTO time between the target primary and standby database instances (unit: second).
current_rpo	int	Flow control information, that is, log RPO time of the current primary and standby database instances (unit: second).
target_rpo	int	Flow control information, that is, the RPO time between the target primary and standby database instances (unit: second).
rto_sleep_time	int	RTO flow control information, that is, the expected sleep time (unit: μs) required by walsender on the host to reach the specified RTO.
rpo_sleep_time	int	RPO flow control information, that is, the expected sleep time (unit: μs) required by xlogInsert on the host to reach the specified RPO.

- gs_hadr_remote_rto_and_rpo_stat()

  Description: Displays the log flow control information of the primary and standby database instances of other non-local data shards in streaming DR mode. This function is not supported in centralized deployment scenarios.

7.6.26.7 Snapshot Synchronization Functions

Snapshot synchronization functions save the current snapshot and return its identifier.

- pg_export_snapshot()

  Description: Saves the current snapshot and returns its identifier.

  Return type: text

  Note: **pg_export_snapshot** saves the current snapshot and returns a text string identifying the snapshot. This string must be passed to clients that want to import the snapshot. A snapshot can be imported when the **set transaction snapshot snapshot_id;** command is executed. Doing so is possible only when the transaction is set to the **SERIALIZABLE** or **REPEATABLE READ** isolation level. GaussDB does not support these two isolation levels currently. The output of the function cannot be used as the input of **set transaction snapshot**.

- pg_export_snapshot_and_csn()

  Description: Saves the current snapshot and returns its identifier. Compared with **pg_export_snapshot()**, **pg_export_snapshot()** returns a CSN, indicating the CSN of the current snapshot.

  Return type: text

7.6.26.8 Database Object Functions

Database Object Size Functions

Database object size functions calculate the actual disk space used by database objects.

- pg_column_size(any)

  Description: Specifies the number of bytes used to store a particular value (possibly compressed).

  Return type: int

  Note: **pg_column_size** displays the space for storing an independent data value.

  ```
  gaussdb=# SELECT pg_column_size(1);
   pg_column_size
  ----------------
               4
  (1 row)
  ```

- pg_database_size(oid)

  Description: Specifies the disk space used by the database with the specified OID.

  Return type: bigint

- pg_database_size(name)

  Description: Specifies the disk space used by the database with the specified name.

  Return type: bigint

  Note: **pg_database_size** receives the OID or name of a database and returns the disk space used by the corresponding object.

Example:

```
gaussdb=# SELECT pg_database_size('testdb');
 pg_database_size
------------------
         51590112
(1 row)
```

- pg_relation_size(oid)

  Description: Specifies the disk space used by the table with a specified OID or index.

  Return type: bigint

- get_db_source_datasize()

  Description: Estimates the total size of non-compressed data in the current database.

  Return type: bigint

  Note: Perform an analysis before this function is called.

  Example:

```
gaussdb=# analyze;
ANALYZE
gaussdb=# SELECT get_db_source_datasize();
 get_db_source_datasize
------------------------
            35384925667
(1 row)
```

- pg_relation_size(text)

  Description: Specifies the disk space used by the table with a specified name or index. The table name can be schema-qualified.

  Return type: bigint

- pg_relation_size(relation regclass, fork text)

  Description: Specifies the disk space used by the specified bifurcating tree ('main', 'fsm', or 'vm') of a certain table or index.

  Return type: bigint

- pg_relation_size(relation regclass)

  Description: Is an abbreviation of **pg_relation_size(..., 'main')**.

  Return type: bigint

  Note: **pg_relation_size** receives the OID or name of a table, an index, or a compressed table, and returns the size.

- pg_partition_size(oid, oid)

  Description: Specifies the disk space used by the partition with a specified OID. The first **oid** is the OID of the table and the second **oid** is the OID of the partition.

  Return type: bigint

- pg_partition_size(text, text)

  Description: Specifies the disk space used by the partition with a specified name. The first **text** is the table name and the second **text** is the partition name.

  Return type: bigint

- pg_partition_indexes_size(oid, oid)

Description: Specifies the disk space used by the index of the partition with a specified OID. The first **oid** is the OID of the table and the second **oid** is the OID of the partition.

Return type: bigint

- pg_partition_indexes_size(text, text)

  Description: Specifies the disk space used by the index of the partition with a specified name. The first **text** is the table name and the second **text** is the partition name.

  Return type: bigint

- pg_indexes_size(regclass)

  Description: Specifies the total disk space used by the index appended to the specified table.

  Return type: bigint

- pg_size_pretty(bigint)

  Description: Converts a size in bytes expressed as a 64-bit integer into a human-readable format with size units.

  Return type: text

- pg_size_pretty(numeric)

  Description: Converts a size in bytes expressed as a numeric value into a human-readable format with size units.

  Return type: text

  Note: **pg_size_pretty** formats the results of other functions into a human-readable format. KB, MB, GB, and TB can be used.

- pg_table_size(regclass)

  Description: Specifies the disk space used by the specified table, excluding indexes (but including TOAST, free space mapping, and visibility mapping).

  Return type: bigint

- pg_tablespace_size(oid)

  Description: Specifies the disk space used by the tablespace with a specified OID.

  Return type: bigint

- pg_tablespace_size(name)

  Description: Specifies the disk space used by the tablespace with a specified name.

  Return type: bigint

  Note:

  pg_tablespace_size receives the OID or name of a database and returns the disk space used by the corresponding object.

- pg_total_relation_size(oid)

  Description: Specifies the disk space used by the table with a specified OID, including the index and the compressed data.

  Return type: bigint

- pg_total_relation_size(regclass)

Description: Specifies the total disk space used by the specified table, including all indexes and TOAST data.

Return type: bigint

- pg_total_relation_size(text)

  Description: Specifies the disk space used by the table with a specified name, including the index and the compressed data. The table name can be schema-qualified.

  Return type: bigint

  Note: **pg_total_relation_size** receives the OID or name of a table or a compressed table, and returns the sizes of the data, related indexes, and the compressed table in bytes.

- datalength(any)

  Description: Specifies the number of bytes used by an expression of a specified data type (data management space, data compression, or data type conversion is not considered).

  Return type: int

  Note: **datalength** is used to calculate the space of an independent data value.

  Example:

```
gaussdb=# SELECT datalength(1);
datalength
------------
4
(1 row)
```

The following table lists the supported data types and calculation methods.

Data Type			Storage Space
Numeric data types	Integer types	TINYINT	1
		SMALLINT	2
		INTEGER	4
		BINARY_INTEGER	4
		BIGINT	8
	Arbitrary precision types	DECIMAL	Every four decimal digits occupy two bytes. The digits before and after the decimal point are calculated separately.
		NUMERIC	Every four decimal digits occupy two bytes. The digits before and after the decimal point are calculated separately.

Data Type			Storage Space
		NUMBER	Every four decimal digits occupy two bytes. The digits before and after the decimal point are calculated separately.
	Sequence integer	SMALLSERIAL	2
		SERIAL	4
		BIGSERIAL	8
		LARGESERIAL	Every four decimal digits occupy two bytes. The digits before and after the decimal point are calculated separately.
	Floating point types	FLOAT4	4
		DOUBLE PRECISION	8
		FLOAT8	8
		BINARY_DOUBLE	8
		FLOAT[(p)]	Every four decimal digits occupy two bytes. The digits before and after the decimal point are calculated separately.
		DEC[(p[,s])]	Every four decimal digits occupy two bytes. The digits before and after the decimal point are calculated separately.
		INTEGER[(p[,s])]	Every four decimal digits occupy two bytes. The digits before and after the decimal point are calculated separately.
Boolean data types	Boolean type	BOOLEAN	1
Character data types	Character types	CHAR	n
		CHAR(n)	n
		CHARACTER(n)	n
		NCHAR(n)	n

Data Type			Storage Space
		VARCHAR(n)	n
		CHARACTER	Actual number of bytes of a character
		VARYING(n)	Actual number of bytes of a character
		VARCHAR2(n)	Actual number of bytes of a character
		NVARCHAR(n)	Actual number of bytes of a character
		NVARCHAR2(n)	Actual number of bytes of a character
		TEXT	Actual number of bytes of a character
		CLOB	Actual number of bytes of a character
Time data types	Time types	DATE	8
		TIME	8
		TIMEZ	12
		TIMESTAMP	8
		TIMESTAMPZ	8
		SMALLDATETIME	8
		INTERVAL DAY TO SECOND	16
		INTERVAL	16
		RELTIME	4
		ABSTIME	4
		TINTERVAL	12

Database Object Position Functions

- pg_relation_filenode(relation regclass)

  Description: Specifies the ID of a filenode with the specified relationship.

  Return type: oid

  Note: **pg_relation_filenode** receives the OID or name of a table, an index, a sequence, or a compressed table, and returns the ID of **filenode** allocated to it. **filenode** is the basic component of the file name used by the relationship.

For most tables, the result is the same as that of **pg_class.relfilenode**. For a specified system directory, **relfilenode** is set to **0** and this function must be used to obtain the correct value. If a relationship that is not stored is transmitted, such as a view, this function returns a null value.

- pg_relation_filepath(relation regclass)

  Description: Specifies the name of a file path with the specified relationship. It can only be used for non-segment-page relationships.

  Return type: text

  Description: **pg_relation_filepath** is similar to **pg_relation_filenode**, except that **pg_relation_filepath** returns the whole file path name (relative to the data directory **PGDATA** of the database) for the relationship.

  For segment-page relationships, you are advised to use functions or views. For example:

  ```
  SELECT e.*, f.file_name
  FROM gs_seg_extents e, gs_seg_datafiles f
  WHERE e.tablespace_name = f.tablespace_name ADN e.bucketnode = f.bucketnode ADN e.file_id =
  f.file_id ADN e.forknum = f.forknum;
  ```

- pg_filenode_relation(tablespace oid, filenode oid)

  Description: Obtains the table names corresponding to the tablespace and relfilenode.

  Return type: regclass

- pg_partition_filenode(partition_oid)

  Description: Obtains **filenode** corresponding to the OID lock of a specified partitioned table.

  Return type: oid

- pg_partition_filepath(partition_oid)

  Description: Specifies the file path name of a specified partition. It can only be used for non-segment-page relationships.

  Return type: text

  Note: For segment-page relationships, you are advised to use functions or views. For example:

  ```
  SELECT e.*, f.file_name
  FROM gs_seg_extents e, gs_seg_datafiles f
  WHERE e.tablespace_name = f.tablespace_name ADN e.bucketnode = f.bucketnode ADN e.file_id =
  f.file_id ADN e.forknum = f.forknum;
  ```

Recycle Bin Object Functions

- gs_is_recycle_object(classid, objid, objname)

  Description: Determines whether an object is in the recycle bin.

  Return type: Boolean

7.6.26.9 Advisory Lock Functions

Advisory lock functions manage advisory locks.

- pg_advisory_lock(key bigint)

  Description: Obtains an exclusive session-level advisory lock.

  Return type: void

Note: **pg_advisory_lock** locks resources defined by an application. The resources can be identified using a 64-bit or two nonoverlapped 32-bit key values. If another session locks the resources, the function blocks the resources until they can be used. The lock is exclusive. Multiple locking requests are pushed into the stack. Therefore, if the same resource is locked three times, it must be unlocked three times so that it is released to another session.

- pg_advisory_lock(key1 int, key2 int)

  Description: Obtains an exclusive session-level advisory lock.

  Return type: void

  Note: Only users with the sysadmin permission can add session-level exclusive advisory locks to the key-value pair (65535, 65535).

- pg_advisory_lock(lock_id int4, lock_id int4, datebase_name Name)

  Description: Obtains the exclusive advisory lock of a specified database by inputting the lock ID and database name.

  Return type: void

- pg_advisory_lock_shared(key bigint)

  Description: Obtains a shared session-level advisory lock.

  Return type: void

- pg_advisory_lock_shared(key1 int, key2 int)

  Description: Obtains a shared session-level advisory lock.

  Return type: void

  Note: **pg_advisory_lock_shared** works in the same way as **pg_advisory_lock**, except that **pg_advisory_lock_shared** obtains an advisory lock shared with other sessions requesting the lock, while **pg_advisory_lock** obtains an exclusive advisory lock.

- pg_advisory_unlock(key bigint)

  Description: Releases an exclusive session-level advisory lock.

  Return type: Boolean

- pg_advisory_unlock(key1 int, key2 int)

  Description: Releases an exclusive session-level advisory lock.

  Return type: Boolean

  Note: **pg_advisory_unlock** releases the obtained exclusive advisory lock. If the release is successful, the function returns **true**. If the lock was not held, it will return **false**. In addition, an SQL warning will be reported by the server.

- pg_advisory_unlock(lock_id int4, lock_id int4, datebase_name Name)

  Description: Releases an exclusive advisory lock of a specified database by inputting the lock ID and database name.

  Return type: Boolean

  Note: If the release is successful, **true** is returned. If no lock is held, **false** is returned.

- pg_advisory_unlock_shared(key bigint)

  Description: Releases a shared session-level advisory lock.

  Return type: Boolean

- pg_advisory_unlock_shared(key1 int, key2 int)

  Description: Releases a shared session-level advisory lock.

  Return type: Boolean

  Note: **pg_advisory_unlock_shared** works in the same way as **pg_advisory_unlock**, except that it releases a shared advisory lock.

- pg_advisory_unlock_all()

  Description: Releases all advisory locks owned by the current session.

  Return type: void

  Note: **pg_advisory_unlock_all** releases all advisory locks owned by the current session. The function is implicitly invoked when the session ends even if the client is abnormally disconnected.

- pg_advisory_xact_lock(key bigint)

  Description: Obtains an exclusive transaction-level advisory lock.

  Return type: void

- pg_advisory_xact_lock(key1 int, key2 int)

  Description: Obtains an exclusive transaction-level advisory lock.

  Return type: void

  Note: **pg_advisory_xact_lock** works in the same way as **pg_advisory_lock**, except that the lock is automatically released at the end of the current transaction and cannot be released explicitly. Only users with the sysadmin permission can add transaction-level exclusive advisory locks to the key-value pair (65535, 65535).

- pg_advisory_xact_lock_shared(key bigint)

  Description: Obtains a shared transaction-level advisory lock.

  Return type: void

- pg_advisory_xact_lock_shared(key1 int, key2 int)

  Description: Obtains a shared transaction-level advisory lock.

  Return type: void

  Note: **pg_advisory_xact_lock_shared** works in the same way as **pg_advisory_lock_shared**, except that the lock is automatically released at the end of the current transaction and cannot be released explicitly.

- pg_try_advisory_lock(key bigint)

  Description: Obtains an exclusive session-level advisory lock if available.

  Return type: Boolean

  Note: **pg_try_advisory_lock** is similar to **pg_advisory_lock**, except that **pg_try_advisory_lock** does not block the resource until the resource is released. It either immediately obtains the lock and returns **true** or returns **false**, which indicates the lock cannot be performed currently.

- pg_try_advisory_lock(key1 int, key2 int)

  Description: Obtains an exclusive session-level advisory lock if available.

  Return type: Boolean

  Note: Only users with the sysadmin permission can add session-level exclusive advisory locks to the key-value pair (65535, 65535).

- pg_try_advisory_lock_shared(key bigint)

  Description: Obtains a shared session-level advisory lock if available.

  Return type: Boolean

- pg_try_advisory_lock_shared(key1 int, key2 int)

  Description: Obtains a shared session-level advisory lock if available.

  Return type: Boolean

  Note: **pg_try_advisory_lock_shared** works in the same way as **pg_try_advisory_lock**, except that **pg_try_advisory_lock_shared** attempts to obtain a shared lock instead of an exclusive lock.

- pg_try_advisory_xact_lock(key bigint)

  Description: Obtains an exclusive transaction-level advisory lock if available.

  Return type: Boolean

- pg_try_advisory_xact_lock(key1 int, key2 int)

  Description: Obtains an exclusive transaction-level advisory lock if available.

  Return type: Boolean

  Note: **pg_try_advisory_xact_lock** works in the same way as **pg_try_advisory_lock**, except that the lock, if acquired, is automatically released at the end of the current transaction and cannot be released explicitly. Only users with the sysadmin permission can add transaction-level exclusive advisory locks to the key-value pair (65535, 65535).

- pg_try_advisory_xact_lock_shared(key bigint)

  Description: Obtains a shared transaction-level advisory lock if available.

  Return type: Boolean

- pg_try_advisory_xact_lock_shared(key1 int, key2 int)

  Description: Obtains a shared transaction-level advisory lock if available.

  Return type: Boolean

  Note: **pg_try_advisory_xact_lock_shared** works in the same way as **pg_try_advisory_lock_shared**, except that the lock, if acquired, is automatically released at the end of the current transaction and cannot be released explicitly.

- lock_cluster_ddl()

  Description: Attempts to obtain a session-level exclusive advisory lock for all active primary database nodes in the database.

  Return type: Boolean

  Note: Only users with the sysadmin permission can call this function.

- unlock_cluster_ddl()

  Description: Attempts to add a session-level exclusive advisory lock on the primary database node.

  Return type: Boolean

7.6.26.10 Logical Replication Functions

NOTE

When using the logical replication functions, set the GUC parameter **wal_level** to **logical**. For details about the configuration, see "Logical Decoding > Logical Decoding by SQL Function Interfaces" in *Feature Guide*.

- pg_create_logical_replication_slot('slot_name', 'plugin_name', 'output_order')

  Description: Creates a logical replication slot.

  Parameter:

  – slot_name

  Indicates the name of the streaming replication slot.

  Value range: a string, supporting only lowercase letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.). One or two periods cannot be used alone as the replication slot name.

  – plugin_name

  Indicates the name of the plug-in.

  Value range: a string, supporting **mppdb_decoding**

  – output_order

  Indicates the output sequence of the replication slot decoding results. This parameter is optional.

  Value range: **0** and **1**. The default value is **0**.

  - **0**: The replication slot decoding results are sorted by transaction COMMIT LSN. In this case, the value of **confirmed_csn** of the replication slot is **0**. This replication slot is called an LSN-based replication slot.
  - **1**: The replication slot decoding results are sorted by transaction CSN. In this case, the value of **confirmed_csn** of the replication slot is a non-zero value. This replication slot is called a CSN-based replication slot.

  Return type: name, text

  Example:

```
gaussdb=# SELECT * FROM pg_create_logical_replication_slot('slot_lsn','mppdb_decoding',0);
 slotname | xlog_position
----------+---------------
 slot_lsn | 0/6D08B58
(1 row)

gaussdb=# SELECT * FROM pg_create_logical_replication_slot('slot_csn','mppdb_decoding',1);
 slotname | xlog_position
----------+---------------
 slot_csn | 0/59AD800
(1 row)
```

  Note: The first return value is the slot name, and the second one has different meanings in LSN-based replication slots and CSN-based replication slots. For an LSN-based replication slot, the value is **confirmed_flush** of the replication slot, indicating that transactions whose commit LSN is less than or equal to the value will not be decoded and output. For a CSN-based replication slot, the value is **confirmed_csn** of the replication slot, indicating that transactions whose CSN is less than or equal to the value will not be decoded and output. Users who call this function must have the SYSADMIN permission or the

REPLICATION permission, or inherit the gs_role_replication permission of the built-in role. Currently, this function can be called only on the primary node.

- pg_create_physical_replication_slot('slot_name', 'isDummyStandby')

  Description: Creates a physical replication slot.

  Parameter:

  – slot_name

    Indicates the name of the streaming replication slot.

    Value range: a string, supporting only lowercase letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.). One or two periods cannot be used alone as the replication slot name.

  – isDummyStandby

    Reserved parameter.

    Type: bool

  Return type: name, text

  NOTE

  - Users who call this function must have the SYSADMIN permission or the REPLICATION permission, or inherit the gs_role_replication permission of the built-in role.
  - The physical replication slot created by this function does not have **restart_lsn**. Therefore, the slot is considered invalid and will be automatically deleted when the checkpoint is performed.

- pg_drop_replication_slot('slot_name')

  Description: Deletes a streaming replication slot.

  Parameter:

  – slot_name

    Indicates the name of the streaming replication slot.

    Value range: a string, supporting only lowercase letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.). One or two periods cannot be used alone as the replication slot name.

  Return type: void

  Note: Users who call this function must have the SYSADMIN permission or the REPLICATION permission, or inherit the gs_role_replication permission of the built-in role. Currently, this function can be called only on the primary node.

- pg_logical_slot_peek_changes('slot_name', 'upto_lsn', upto_nchanges, 'options_name', 'options_value')

  Description: Performs decoding but does not go to the next streaming replication slot. (The decoded result will be returned again during the next decoding.)

  Parameter:

  – slot_name

    Indicates the name of the streaming replication slot.

    Value range: a string, supporting only lowercase letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.). One or two periods cannot be used alone as the replication slot name.

- upto_lsn

  For the CSN-based logical replication slot, the decoding is complete until the transaction whose CSN is less than or equal to the value is decoded (a transaction whose CSN is greater than the specified CSN may be decoded). For the LSN-based replication slot, the decoding is complete until the first transaction whose COMMIT LSN is greater than or equal to the value is decoded.

  Value range: a string, for example, '1/2AAFC60', '0/A060', or '3A/0' (a hexadecimal uint64 value containing two uint32 values separated by a slash (/); if any uint32 value is **0**, **0** is displayed.) **NULL** indicates that the end position of decoding is not specified.

- upto_nchanges

  Indicates the number of decoded records (including the **begin** and **commit** timestamps). Assume that there are three transactions, which involve 3, 5, and 7 records, respectively. If **upto_nchanges** is set to **4**, 8 records of the first two transactions will be decoded. Specifically, decoding is stopped when the number of decoded records exceeds the value of **upto_nchanges** after decoding in the first two transactions is finished.

  Value range: a non-negative integer

  NOTE

  If any of the **upto_lsn** and **upto_nchanges** values is reached, decoding ends.

- **options**: Specifies optional parameters, consisting of **options_name** and **options_value**.

  - include-xids

    Specifies whether the decoded **data** column contains XID information.

    Value range: Boolean. The default value is **true**.

    - **false**: The decoded **data** column does not contain XID information.
    - **true**: The decoded **data** column contains XID information.

  - skip-empty-xacts

    Specifies whether to ignore empty transaction information during decoding.

    Value range: Boolean. The default value is **false**.

    - **false**: The empty transaction information is not ignored during decoding.
    - **true**: The empty transaction information is ignored during decoding.

  - include-timestamp

    Specifies whether decoded information contains the **commit** timestamp.

    Value range: Boolean. The default value is **true**.

    - **false**: The decoded information does not contain the **commit** timestamp.

- **true**: The decoded information contains the **commit** timestamp.

- only-local

  Specifies whether to decode only local logs.

  Value range: Boolean. The default value is **true**.

  - **false**: Non-local logs and local logs are decoded.
  - **true**: Only local logs are decoded.

- force-binary

  Specifies whether to output the decoding result in binary format.

  Value range: Boolean. The default value is **false**.

  - **false**: The decoding result is output in text format.
  - The value cannot be set to **true** currently.

- white-table-list

  Whitelist parameter, including the schema and table name to be decoded.

  Value range: a string that contains table names in the whitelist. Different tables are separated by commas (,). An asterisk (*) is used to fuzzily match all tables. Schema names and table names are separated by periods (.). No space character is allowed. The following is an example:

  ```
  SELECT * FROM pg_logical_slot_peek_changes('slot1', NULL, 4096, 'white-table-list', 'public.t1,public.t2');
  ```

- max-txn-in-memory

  Memory control parameter. The unit is MB. If the memory occupied by a single transaction is greater than the value of this parameter, data is flushed to disks.

  Value range: an integer ranging from 0 to 100. The default value is **0**, indicating that memory control is disabled.

- max-reorderbuffer-in-memory

  Memory control parameter. The unit is GB. If the total memory (including the cache) of transactions being concatenated in the sender thread is greater than the value of this parameter, the current decoding transaction is flushed to disks.

  Value range: an integer ranging from 0 to 100. The default value is **0**, indicating that memory control is disabled.

- include-user

  Specifies whether the BEGIN logical log of a transaction records the username of the transaction.

  Value range: Boolean. The default value is **false**.

  - **false**: The BEGIN logical log of a transaction does not record the username of the transaction.
  - **true**: The BEGIN logical log of a transaction records the username of the transaction.

- exclude-userids

  Specifies the OID of a blacklisted user.

  Value range: OIDs of blacklisted users. Multiple OIDs are separated by commas (,). The system does not check whether the OIDs exist.

- exclude-users

  Specifies the name of a blacklisted user.

  Value range: names of blacklisted users. Multiple names are separated by commas (,). **dynamic-resolution** specifies whether to dynamically parse and identify usernames. If the decoding is interrupted because the user does not exist and the corresponding blacklisted user does not exist at the time when logs are generated, you can set **dynamic-resolution** to **true** or delete the username from the blacklist to start decoding and continue to obtain logical logs.

- dynamic-resolution

  Specifies whether to dynamically parse the names of blacklisted users.

  Value range: Boolean. The default value is **true**.

  - **false**: An error is reported and the logical decoding exits when the decoding detects that a user does not exist in blacklist **exclude-users**.
  - **true**: Decoding continues when it detects that a user does not exist in blacklist **exclude-users**.

- enable-ddl-decoding

  Specifies the DDL statement reverse parsing process and output format for logical decoding.

  Value range: Boolean. The default value is **false**.

  - **false**: Logical decoding of DDL statements is disabled.
  - **true**: Logical decoding of DDL statements is enabled.

- enable-ddl-json-format

  Specifies the DDL statement reverse parsing process and output format for logical decoding.

  Value range: Boolean. The default value is **false**.

  - **false**: The DDL statement reverse parsing result is output in text format.
  - **true**: The DDL statement reverse parsing result is output in JSON format.

NOTE

For details about other configuration options, see "Logical Replication > Logical Decoding > Logical Decoding Options" in the *Feature Guide*.

Return type: text, xid, text

Example:

```
gaussdb=# SELECT * FROM pg_logical_slot_peek_changes('slot_lsn',NULL,4096,'skip-empty-xacts','on');
 location | xid |                                                data
```



```
-----------+-------
+-----------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------
 0/6D0B500 | 46914 | BEGIN 46914
 0/6D0B530 | 46914 | {"table_name":"public.t1","op_type":"INSERT","columns_name":
["a","b"],"columns_type":["integer","integer"],"columns_val":["3","1"],"old_keys_name":
[],"old_keys_type":[],"old_keys_val":[]}
 0/6D0B8B8 | 46914 | COMMIT 46914 (at 2023-02-22 17:29:31.090018+08) CSN 94034528
 0/6D0BB58 | 46915 | BEGIN 46915
 0/6D0BB88 | 46915 | {"table_name":"public.t1","op_type":"INSERT","columns_name":
["a","b"],"columns_type":["integer","integer"],"columns_val":["3","2"],"old_keys_name":
[],"old_keys_type":[],"old_keys_val":[]}
 0/6D0BF08 | 46915 | COMMIT 46915 (at 2023-02-22 17:31:30.672093+08) CSN 94034568
 0/6D0BF08 | 46916 | BEGIN 46916
 0/6D0BF38 | 46916 | {"table_name":"public.t1","op_type":"INSERT","columns_name":
["a","b"],"columns_type":["integer","integer"],"columns_val":["3","3"],"old_keys_name":
[],"old_keys_type":[],"old_keys_val":[]}
 0/6D0C218 | 46916 | COMMIT 46916 (at 2023-02-22 17:31:34.438319+08) CSN 94034570
(9 rows)

gaussdb=# SELECT * FROM pg_logical_slot_peek_changes('slot_csn',NULL,4096,'skip-empty-xacts','on');
 location  |  xid  |                                                                                                                                data
-----------+-------
+-----------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------
 0/0       | 46914 | BEGIN CSN: 94034528
 0/0       | 46914 | {"table_name":"public.t1","op_type":"INSERT","columns_name":
["a","b"],"columns_type":["integer","integer"],"columns_val":["3","1"],"old_keys_name":
[],"old_keys_type":[],"old_keys_val":[]}
 0/59ADA60 | 46914 | COMMIT 46914 (at 2023-02-22 17:29:31.090018+08) CSN 94034528
 0/59ADA60 | 46915 | BEGIN CSN: 94034568
 0/59ADA60 | 46915 | {"table_name":"public.t1","op_type":"INSERT","columns_name":
["a","b"],"columns_type":["integer","integer"],"columns_val":["3","2"],"old_keys_name":
[],"old_keys_type":[],"old_keys_val":[]}
 0/59ADA88 | 46915 | COMMIT 46915 (at 2023-02-22 17:31:30.672093+08) CSN 94034568
 0/59ADA88 | 46916 | BEGIN CSN: 94034570
 0/59ADA88 | 46916 | {"table_name":"public.t1","op_type":"INSERT","columns_name":
["a","b"],"columns_type":["integer","integer"],"columns_val":["3","3"],"old_keys_name":
[],"old_keys_type":[],"old_keys_val":[]}
 0/59ADA8A | 46916 | COMMIT 46916 (at 2023-02-22 17:31:34.438319+08) CSN 94034570
(9 rows)
```

Note: The decoding result returned by the function contains three columns, corresponding to the preceding return value types, which are the LSN (for an LSN-based replication slot) or CSN (for a CSN-based replication slot), XID, and decoded content, respectively. If the **location** column indicates the CSN, the value of the **location** column is updated only when the commit logs are decoded.

Users who call this function must have the SYSADMIN permission or the REPLICATION permission, or inherit the gs_role_replication permission of the built-in role.

- pg_logical_slot_get_changes('slot_name', 'upto_lsn', upto_nchanges, 'options_name', 'options_value')

  Description: Performs decoding and goes to the next streaming replication slot.

  Parameter: This function has the same parameters as **pg_logical_slot_peek_changes**. For details, see **pg_logical_slot_peek_changes**.

  Note: Users who call this function must have the SYSADMIN permission or the REPLICATION permission, or inherit the gs_role_replication permission of the built-in role. This function can be called on the primary or standby node.

If this function is called on the standby node, the corresponding logical replication slot number on the primary node is updated.

NOTE

If this function is executed on the standby node, a WAL sender of the primary node is occupied when the replication slot number on the primary node is updated. The logical decoding function reserves a WAL sender for each logical replication slot. Therefore, if this function is executed in normal scenarios, the logical replication slot number on the primary node is updated normally. If this function is executed continuously in a short period of time, the primary node fails to update the slot number and no error is reported.

- pg_logical_slot_peek_binary_changes('slot_name', 'upto_lsn', upto_nchanges, 'options_name', 'options_value')

  Description: Performs decoding in binary mode and does not go to the next streaming replication slot. (The decoded data can be obtained again during the next decoding.)

  Parameter:

  – slot_name

    Indicates the name of the streaming replication slot.

    Value range: a string, supporting only lowercase letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.). One or two periods cannot be used alone as the replication slot name.

  – upto_lsn

    For the CSN-based logical replication slot, the decoding is complete until the transaction whose CSN is less than or equal to the value is decoded (a transaction whose CSN is greater than the specified CSN may be decoded). For the LSN-based replication slot, the decoding is complete until the first transaction whose COMMIT LSN is greater than or equal to the value is decoded.

    Value range: a string, for example, '1/2AAFC60', '0/A060', or '3A/0' (a hexadecimal uint64 value containing two uint32 values separated by a slash (/); if any uint32 value is **0**, **0** is displayed.) **NULL** indicates that the end position of decoding is not specified.

  – upto_nchanges

    Indicates the number of decoded records (including the **begin** and **commit** timestamps). Assume that there are three transactions, which involve 3, 5, and 7 records, respectively. If **upto_nchanges** is set to **4**, 8 records of the first two transactions will be decoded. Specifically, decoding is stopped when the number of decoded records exceeds the value of **upto_nchanges** after decoding in the first two transactions is finished.

    Value range: a non-negative integer

    NOTE

    If any of the **upto_lsn** and **upto_nchanges** values is reached, decoding ends.

  – **options**: Specifies optional parameters, consisting of **options_name** and **options_value**.

    ▪ include-xids

Specifies whether the decoded **data** column contains XID information.

Value range: Boolean. The default value is **true**.

- **false**: The decoded **data** column does not contain XID information.
- **true**: The decoded **data** column contains XID information.

- skip-empty-xacts

  Specifies whether to ignore empty transaction information during decoding.

  Value range: Boolean. The default value is **false**.

  - **false**: The empty transaction information is not ignored during decoding.
  - **true**: The empty transaction information is ignored during decoding.

- include-timestamp

  Specifies whether decoded information contains the **commit** timestamp.

  Value range: Boolean. The default value is **true**.

  - **false**: The decoded information does not contain the **commit** timestamp.
  - **true**: The decoded information contains the **commit** timestamp.

- only-local

  Specifies whether to decode only local logs.

  Value range: Boolean. The default value is **true**.

  - **false**: Non-local logs and local logs are decoded.
  - **true**: Only local logs are decoded.

- force-binary

  This parameter is useless for the function.

  Value range: Boolean. The result is output in binary format.

- white-table-list

  Whitelist parameter, including the schema and table name to be decoded.

  Value range: a string that contains table names in the whitelist. Different tables are separated by commas (,). An asterisk (*) is used to fuzzily match all tables. Schema names and table names are separated by periods (.). No space character is allowed. Example: **select * from pg_logical_slot_peek_binary_changes('slot1', NULL, 4096, 'white-table-list', 'public.t1,public.t2');**

- max-txn-in-memory

  Memory control parameter. The unit is MB. If the memory occupied by a single transaction is greater than the value of this parameter, data is flushed to disks.

Value range: an integer ranging from 0 to 100. The default value is **0**, indicating that memory control is disabled.

- max-reorderbuffer-in-memory

  Memory control parameter. The unit is GB. If the total memory (including the cache) of transactions being concatenated in the sender thread is greater than the value of this parameter, the current decoding transaction is flushed to disks.

  Value range: an integer ranging from 0 to 100. The default value is **0**, indicating that memory control is disabled.

- include-user

  Specifies whether the BEGIN logical log of a transaction records the username of the transaction.

  Value range: Boolean. The default value is **false**.

  - **false**: The BEGIN logical log of a transaction does not record the username of the transaction.
  - **true**: The BEGIN logical log of a transaction records the username of the transaction.

- exclude-userids

  Specifies the OID of a blacklisted user.

  Value range: OIDs of blacklisted users. Multiple OIDs are separated by commas (,). The system does not check whether the OIDs exist.

- exclude-users

  Specifies the name of a blacklisted user.

  Value range: names of blacklisted users. Multiple names are separated by commas (,). **dynamic-resolution** specifies whether to dynamically parse and identify usernames. If the decoding is interrupted because the user does not exist and the corresponding blacklisted user does not exist at the time when logs are generated, you can set **dynamic-resolution** to **true** or delete the username from the blacklist to start decoding and continue to obtain logical logs.

- dynamic-resolution

  Specifies whether to dynamically parse the names of blacklisted users.

  Value range: Boolean. The default value is **true**.

  - **false**: An error is reported and the logical decoding exits when the decoding detects that a user does not exist in blacklist **exclude-users**.
  - **true**: Decoding continues when it detects that a user does not exist in blacklist **exclude-users**.

NOTE

Some configuration options do not take effect even if they are configured in functions. For details, see "Logical Replication > Logical Decoding > Logical Decoding Options" in the *Feature Guide*.

Return type: text, xid, bytea

Note: The function returns the decoding result. Each decoding result contains three columns, corresponding to the above return types and indicating the LSN position, XID, and decoded content in binary format, respectively. Users who call this function must have the SYSADMIN permission or the REPLICATION permission, or inherit the gs_role_replication permission of the built-in role.

- pg_logical_slot_get_binary_changes('slot_name', 'upto_lsn', upto_nchanges, 'options_name', 'options_value')

  Description: Performs decoding in binary mode and does not go to the next streaming replication slot.

  Parameter: This function has the same parameters as **pg_logical_slot_peek_binary_changes**. For details, see **•pg_logical_slot_peek_bi...**

  Note: Users who call this function must have the SYSADMIN permission or the REPLICATION permission, or inherit the gs_role_replication permission of the built-in role. Currently, this function can be called only on the primary node.

- pg_replication_slot_advance ('slot_name', 'upto_lsn')

  Description: Directly goes to the streaming replication slot for a specified **upto_lsn**, without outputting any decoded result.

  Parameter:

  – slot_name

    Indicates the name of the streaming replication slot.

    Value range: a string, supporting only lowercase letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.). One or two periods cannot be used alone as the replication slot name.

  – upto_lsn

    For the CSN-based logical replication slot, it indicates the target CSN before which logs are decoded. During the next decoding, only the transaction results whose CSN is greater than this value will be output. If the input CSN is smaller than the value of **confirmed_csn** recorded in the current streaming replication slot, the function directly returns the decoded result. If the input CSN is greater than the latest CSN that can be obtained or no CSN is input, the latest CSN will be used for decoding.

    For the LSN-based logical replication slot, it indicates the target LSN before which logs are decoded. During the next decoding, only the transaction results whose LSN is greater than this value will be output. If the input LSN is smaller than the position recorded in the current streaming replication slot, an error is reported. If the input LSN is greater than the LSN of the current physical log or no LSN is input, the latest LSN will be directly used for decoding.

    Value range: a string, for example, **'1/2AAFC60'**, **'0/A060'**, or **'3A/0'** (a hexadecimal uint64 value containing two uint32 values separated by a slash (/); if any uint32 value is **0**, **0** is displayed.) **NULL** indicates that the end position of decoding is not specified.

  Return type: name, text

  Note: A return result contains the slot name and LSN or CSN that is actually used for decoding. Users who call this function must have the SYSADMIN

permission or the REPLICATION permission, or inherit the gs_role_replication permission of the built-in role. This function can be called on the primary or standby node. When this function is called on the standby node, the corresponding logical replication slot number on the primary node is updated.

NOTE

This function can be executed on the standby node to synchronously update the corresponding logical replication slot number on the primary node. If this function is executed on the standby node, a WAL sender of the primary node is occupied when the replication slot number on the primary node is updated. The logical decoding function reserves a WAL sender for each logical replication slot. Therefore, if this function is executed in normal scenarios, the logical replication slot number on the primary node is updated normally. If this function is executed continuously in a short period of time, the primary node fails to update the slot number and no error is reported.

- pg_logical_get_area_changes('LSN_start', 'LSN_end', upto_nchanges, 'decoding_plugin', 'xlog_path', 'options_name', 'options_value')

  Description: Specifies an LSN range or an Xlog file for decoding when no DDL operation is performed.

NOTE

The constraints are as follows:

- The current network and hardware environment are normal.
- It is recommended that the size of a single tuple be less than or equal to 500 MB. If the size ranges from 500 MB to 1 GB, an error is reported.
- Data page replication is not supported for data retrieval that does not fall into Xlogs.
- When the API is called, only when **wal_level** is set to **logical**, the generated log files can be parsed. If the used Xlog file is not of the logical level, the decoded content does not have the corresponding value and type, and there is no other impact. If **wal_level** is not set to **logical**, an error is reported and decoding is not performed.
- The Xlog file can be parsed only by a copy of a completely homogeneous DN, and no DDL operation or VACUUM FULL occurs in the database to ensure that the metadata corresponding to the data can be found.
- Do not read too many Xlog files at a time. If no file is specified for decoding within a specified range, you are advised to read one Xlog file each time. Generally, the memory occupied by an Xlog file during decoding is about two to three times the size of the Xlog file.
- Data before VACUUM FULL cannot be retrieved.
- The Xlog file before scale-out cannot be decoded.
- To decode the UPDATE statement, the table must have a primary key. Otherwise, the WHERE clause in the UPDATE statement is empty.
- Fields of the TOAST, CLOB, or BLOB type cannot be decoded. If a field of the TOAST, CLOB, or BLOB type is decoded, the field is skipped or an error is reported.
- In this decoding mode, the content that can be decoded is decoded based on the Xlog text record data, and the decoding is not performed based on transactions. Therefore, data that is not in the Xlog cannot be decoded.
- If no decoding file is specified from the decoding point, the system checks whether DDL occurs between the decoding start point and the latest redo value. If DDL occurs, the system does not decode all data. If a decoding file is specified, the system checks whether DDL occurs between the start point of the decoding file and the last readable content of the file and between the start point of the Xlog in the data directory and the latest redo value. If a DDL operation is detected, the system does not decode all tables.
- The CSN-based replication slot is not supported.

Note: When separation-of-duty is enabled, only the initial database user can call the function. When separation-of-duty is disabled, the system administrator permission is required.

Parameter:

– LSN_start

  Specifies the LSN at the start of decoding.

  Value range: a string, in the format of *xlogid*/*xrecoff*, for example, **'1/2AAFC60'**. **NULL** indicates that the start position of decoding logs is not limited.

– LSN_end

  Specifies the LSN at the end of decoding.

  Value range: a string, in the format of *xlogid*/*xrecoff*, for example, **'1/2AAFC60'**. **NULL** indicates that the end position of decoding is not specified.

– upto_nchanges

Indicates the number of decoded records (including the **begin** and **commit** timestamps). Assume that there are three transactions, which involve 3, 5, and 7 records, respectively. If **upto_nchanges** is set to **4**, 8 records of the first two transactions will be decoded. Specifically, decoding is stopped when the number of decoded records exceeds the value of **upto_nchanges** after decoding in the first two transactions is finished.

Value range: a non-negative integer

NOTE

If any of the **LSN** and **upto_nchanges** values are reached, decoding ends.

– decoding_plugin

Decoding plug-in, which is a .so plug-in that specifies the output format of the decoded content.

Value range: **mppdb_decoding** and **sql_decoding**.

– xlog_path

Decoding plug-in, which specifies the Xlog absolute path and file level of the decoding file.

Value range: **NULL** or a character string of the absolute path of the Xlog file.

– **options**: This parameter is optional and consists of multiple pairs of **options_name** and **options_value**. You can retain the default values.

- include-xids

  Specifies whether the decoded **data** column contains XID information.

  Value range: Boolean. The default value is **true**.

  false: The decoded **data** column does not contain XID information.

  true: The decoded **data** column contains XID information.

- skip-empty-xacts

  Specifies whether to ignore empty transaction information during decoding.

  Value range: Boolean. The default value is **false**.

  false: The empty transaction information is not ignored during decoding.

  true: The empty transaction information is ignored during decoding.

- include-timestamp

  Specifies whether decoded information contains the **commit** timestamp.

  Value range: Boolean. The default value is **true**.

  false: The decoded information does not contain the **commit** timestamp.

  true: The decoded information contains the **commit** timestamp.

- only-local

Specifies whether to decode only local logs.

Value range: Boolean. The default value is **true**.

false: Non-local logs and local logs are decoded.

true: Only local logs are decoded.

- force-binary

  Specifies whether to output the decoding result in binary format.

  Value range: Boolean. The default value is **false**.

  false: The decoding result is output in text format.

  The value cannot be set to **true** currently.

- white-table-list

  Whitelist parameter, including the schema and table name to be decoded. Value range: a string that contains table names in the whitelist. Different tables are separated by commas (,). An asterisk (*) is used to fuzzily match all tables. Schema names and table names are separated by periods (.). No space character is allowed.

Example:

```
gaussdb=# CREATE TABLE t1(a int, b int);
CREATE TABLE
gaussdb=# SELECT pg_current_xlog_location();
 pg_current_xlog_location
--------------------------
 0/5ECBCD48
(1 row)

gaussdb=# INSERT INTO t1 VALUES(1,1);
INSERT 0 1
gaussdb=# UPDATE t1 SET b = 2 WHERE a = 1;
UPDATE 1
gaussdb=# DELETE FROM t1;
DELETE 1
gaussdb=# SELECT * FROM pg_logical_get_area_changes('0/5ECBCD48', NULL, NULL, 'sql_decoding',
NULL);
  location  |  xid  |                                    data
------------+-------+-----------------------------------------------------------------------------------------
 0/5ECBCD78 | 70718 | insert into public.t1 values (1, 1);
 0/5ECBCEA0 | 70718 | COMMIT 70718 (at 2023-11-01 10:58:51.448885+08) 39319
 0/5ECBCED0 | 70719 | delete from public.t1 where a = 1 and b = 1;insert into public.t1 values (1, 2);
 0/5ECBD028 | 70719 | COMMIT 70719 (at 2023-11-01 10:58:56.487796+08) 39320
 0/5ECBD210 | 70720 | delete from public.t1 where a = 1 and b = 2;
 0/5ECBD338 | 70720 | COMMIT 70720 (at 2023-11-01 10:58:58.856661+08) 39321
(6 rows)

-- For a table with generated columns:
gaussdb=# CREATE TABLE t2(a int, b int GENERATED ALWAYS AS (a + 1) STORED);
CREATE TABLE
gaussdb=# SELECT pg_current_xlog_location();
 pg_current_xlog_location
--------------------------
 0/5F62CFE8
(1 row)

gaussdb=# INSERT INTO t2(a) VALUES(1);
INSERT 0 1
gaussdb=# UPDATE t2 set a = 2 where a = 1;
UPDATE 1
gaussdb=# DELETE FROM t2;
DELETE 1
gaussdb=# SELECT * FROM pg_logical_get_area_changes('0/5F62CFE8', NULL, NULL, 'sql_decoding',
NULL, 'skip-generated-columns', 'on');
```



```
 location  |  xid  |                          data
------------+-------+--------------------------------------------------------------------------
 0/5F62D0C8 | 71293 | insert into public.t2 values (1);
 0/5F62D1F0 | 71293 | COMMIT 71293 (at 2023-11-01 11:11:49.452044+08) 39516
 0/5F62D220 | 71294 | delete from public.t2 where a = 1;insert into public.t2 values (2);
 0/5F62D378 | 71294 | COMMIT 71294 (at 2023-11-01 11:11:54.327701+08) 39517
 0/5F62D408 | 71295 | delete from public.t2 where a = 2;
 0/5F62D530 | 71295 | COMMIT 71295 (at 2023-11-01 11:11:58.362057+08) 39518
(6 rows)
```

- pg_get_replication_slots()

  Description: Obtains the replication slot list.

  Return type: text, text, text, oid, Boolean, xid, xid, text, Boolean, text, xid

  Example:

```
gaussdb=# SELECT * FROM pg_get_replication_slots();
 slot_name |     plugin     | slot_type | datoid | active | xmin | catalog_xmin | restart_lsn |
dummy_standby | confirmed_flush | confirmed_csn
-----------+----------------+-----------+--------+--------+------+--------------+-------------+---------------
+-----------------+---------------
 dn_6002   |                | physical  |      0 | t      |      |              | 0/3622B528  | f             |               |
 dn_6003   |                | physical  |      0 | t      |      |              | 0/3622B528  | f             |               |
 slot_lsn  | mppdb_decoding | logical   | 131072 | f      |      |        66658 | 0/36252350  | f             |
0/362523D0      |
 slot_test | mppdb_decoding | logical   | 131072 | f      |      |        66658 | 0/36251718  | f
|               |      10025527
(4 rows)
```

  Note: In the return values, **slot_name** indicates the name of the replication slot, **plugin** indicates the name of the output plug-in corresponding to the logical replication slot, **slot_type** indicates the type of the replication slot (**physical** indicates the physical replication slot, and **logical** indicates the logical replication slot), **datoid** indicates the OID of the database where the replication slot resides, **active** specifies whether the replication slot is activated (**f**: not activated; **t**: activated), **xmin** indicates the transaction ID of the earliest transaction reserved by the database for the replication slot, **catalog_xmin** indicates the transaction ID of the earliest transaction related to the system catalog reserved by the database for the logical replication slot, **restart_lsn** indicates the physical location of the earliest Xlog required by the replication slot, **dummy_standby** is a reserved parameter, **confirmed_csn** indicates that the client confirms the CSN corresponding to the last transaction in the received log (This parameter is dedicated to logical replication slots.), and **confirmed_flush** indicates that the client confirms the location of the received log (dedicated for logical replication slots).

  NOTICE

  When a query is performed on a DN, the **confirmed_csn** query result of the LSN-based logical replication slot is empty, and the **confirmed_flush** query result of the CSN-based logical replication slot is empty.

- gs_get_parallel_decode_status()

  Description: Monitors the length of the read log queue and decoding result queue of each decoder thread to locate the concurrent decoding performance bottleneck.

  Return type: text, int, text, text, text, int64, int64, TimestampTz

  Example:


```
gaussdb=# select * from gs_get_parallel_decode_status();
 slot_name | parallel_decode_num | read_change_queue_length  | decode_change_queue_length |
reader_lsn | working_txn_cnt | working_txn_memory |       decoded_time
-----------+---------------------+---------------------------+----------------------------+------------
+-----------------+--------------------+-----------------------
 slot1     |                   2 | queue0: 1005, queue1: 320 | queue0: 63, queue1: 748    | 0/1DCE2578
|              42 |          192927504 | 2023-01-10 11:18:22+08
(1 row)
```

Note: In the return values, **slot_name** indicates the replication slot name, **parallel_decode_num** indicates the number of parallel decoder threads in the replication slot, **read_change_queue_length** indicates the current length of the log queue read by each decoder thread, **decode_change_queue_length** indicates the current length of the decoding result queue of each decoder thread, **reader_lsn** indicates the log location read by the reader thread, **working_txn_cnt** indicates the number of transactions being concatenated in the current sender thread, **decoded_time** indicates the time of the latest WAL decoded by the replication slot, and **working_txn_memory** indicates the total memory (in bytes) occupied by the concatenation transactions in the sender thread.

NOTICE

The value of **decoded_time** comes from checkpoint logs and transaction commit logs, which has a certain error. If no log containing the time is decoded, "2000-01-01 08:00:00+08" (depending on the time zone set in the database) is displayed.

- gs_get_slot_decoded_wal_time(slot_name)

  Description: Queries the time of the latest WAL decoded by a replication slot.

  Parameter:

  – slot_name

    Specifies the name of the replication slot to be queried.

    Value range: a string, supporting only letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.).

  Example:

```
gaussdb=# SELECT * FROM gs_get_slot_decoded_wal_time('replication_slot');
 gs_get_slot_decoded_wal_time
------------------------------
 2023-01-10 11:25:22+08
(1 row)
```

  Note: The returned values indicate the time of the latest WAL decoded by the replication slot.

NOTICE

The returned time comes from checkpoint logs and transaction commit logs, which has a certain error. If no log containing the time is decoded, "2000-01-01 08:00:00+08" (depending on the time zone set in the database) is displayed. When you query the latest decoded WAL log time of a logical replication slot that does not exist, **NULL** is returned. In gsql, the display of **NULL** is related to the setting, which can be set using **\pset null'null'**.

- gs_logical_parallel_decode_status('slot_name')

  Description: Obtains the decoding statistics of a replication slot for parallel logical decoding, including 26 rows of statistical items.

  The descriptions of the statistical items are listed in the following table.

  ```
  Record - (stat_id int, stat_name TEXT, value TEXT)
  ```

  Table 7-94 Description

Statistical Item	Description
slot_name	Name of the logical replication slot.
reader_lsn	Location of the logical logs to be decoded.
wal_read_total_time	Time required for loading the log module.
wal_wait_total_time	Time required for waiting for log decoding.
parser_total_time	Processing duration of the reader thread.
decoder_total_time	Processing duration of all decoder threads.
sender_total_time	Processing duration of the sender thread.
net_send_total_time	Time required for the network to send logical logs.
net_wait_total_time	Time required for the network to wait for sending logical logs.
net_send_total_bytes	Number of logical log bytes sent by the network.
transaction_count	Number of transactions.
big_transaction_count	Number of large transactions.
max_transaction_tuples	Maximum number of transaction operation tuples.
sent_transaction_count	Number of transactions sent (by the local database).
spill_disk_transaction_count	Number of flushed transactions.
spill_disk_bytes	Total number of bytes flushed to disks.
spill_disk_count	Number of disk flushing times.
input_queue_full_count	Total number of times that the input queues of all decoder threads are full.
output_queue_full_count	Total number of times that the output queues of all decoder threads are full.

Statistical Item	Description
dml_count	Total number of DML statements in WALs decoded by each decoder thread in the local database.
dml_filtered_count	Total number of DML statements in WALs decoded and filtered by each decoder thread in the local database.
toast_count	Number of modified TOAST table rows.
candidate_catalog_xmin	Indicates the **catalog_xmin** candidate point of the current logical replication slot.
candidate_xmin_lsn	Updates the log confirmation receiving point required by **catalog_xmin**.
candidate_restart_valid	Updates the log confirmation receiving point required by **restart_lsn**.
candidate_restart_lsn	Indicates the **restart_lsn** candidate point of the current logical replication slot.

Parameter:

– slot_name

  Indicates the name of the streaming replication slot.

  Value range: a string, supporting only lowercase letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.). One or two periods cannot be used alone as the replication slot name.

Return type: int, text, text

Example:

```
gaussdb=# SELECT * FROM gs_logical_parallel_decode_status('replication_slot');
 stat_id |           stat_name            |      value
---------+--------------------------------+------------------
       1 | slot_name                      | replication_slot
       2 | reader_lsn                     | 0/357E180
       3 | wal_read_total_time            | 266694599
       4 | wal_wait_total_time            | 266691307
       5 | parser_total_time              | 39971
       6 | decoder_total_time             | 81216
       7 | sender_total_time              | 48193
       8 | net_send_total_time            | 19388
       9 | net_wait_total_time            | 0
      10 | net_send_total_bytes           | 266897
      11 | transaction_count              | 7
      12 | big_transaction_count          | 1
      13 | max_transaction_tuples         | 4096
      14 | sent_transaction_count         | 7
      15 | spill_disk_transaction_count | 1
      16 | spill_disk_bytes               | 244653
      17 | spill_disk_count               | 4096
      18 | input_queue_full_count         | 0
      19 | output_queue_full_count        | 0
      20 | dml_count                      | 4097
      21 | dml_filtered_count             | 0
      22 | toast_count                    | 0
```



```
    23 | candidate_catalog_xmin       | 17152
    24 | candidate_xmin_lsn           | 0/420A598
    25 | candidate_restart_valid      | 0/420A598
    26 | candidate_restart_lsn        | 0/420A598
(26 rows)
```

Note: According to the definitions of statistical items, they must meet the following requirements:

wal_read_total_time >= wal_wait_total_time;

transaction_count >= big_transaction_count;

transaction_count >= sent_transaction_count;

transaction_count >= spill_disk_transaction_count;

dml_count >= dml_filtered_count;

dml_count >= toast_count;

If spill_transaction_count == 0, then spill_disk_bytes == 0;

However, frequent locking and unlocking are required, which greatly affects the performance. As a result, the preceding constraints may not be met in extreme cases.

transaction_count indicates the number of transactions in all databases.

sent_transaction_count indicates the number of transactions sent by the local database because transactions that are not in the local database will not be sent.

If the value of **slot_name** does not exist, the function does not report an error and the return value is empty.

- gs_logical_parallel_decode_reset_status('slot_name')

  Description: Resets indicators in **gs_logical_parallel_decode_status('slot_name')**.

  Parameter:

  – slot_name

    Indicates the name of the streaming replication slot.

    Value range: a string, supporting only lowercase letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.). One or two periods cannot be used alone as the replication slot name.

  Return type: text

  Example:

```
gaussdb=# SELECT * FROM gs_logical_parallel_decode_reset_status('replication_slot');
 gs_logical_parallel_decode_reset_status
-----------------------------------------
 OK
(1 row)

gaussdb=# SELECT * FROM gs_logical_parallel_decode_status('replication_slot');
 stat_id |          stat_name           |      value
---------+------------------------------+------------------
       1 | slot_name                    | replication_slot
       2 | reader_lsn                   | 0/357E420
       3 | wal_read_total_time          | 0
       4 | wal_wait_total_time          | 0
       5 | parser_total_time            | 0
       6 | decoder_total_time           | 0
       7 | sender_total_time            | 0
       8 | net_send_total_time          | 0
       9 | net_wait_total_time          | 0
```



```
    10 | net_send_total_bytes          | 0
    11 | transaction_count             | 0
    12 | big_transaction_count         | 0
    13 | max_transaction_tuples        | 0
    14 | sent_transaction_count        | 0
    15 | spill_disk_transaction_count | 0
    16 | spill_disk_bytes              | 0
    17 | spill_disk_count              | 0
    18 | input_queue_full_count        | 0
    19 | output_queue_full_count       | 0
    20 | dml_count                     | 0
    21 | dml_filtered_count            | 0
    22 | toast_count                   | 0
    23 | candidate_catalog_xmin        | 0
    24 | candidate_xmin_lsn            | 0/0
    25 | candidate_restart_valid       | 0/420A598
    26 | candidate_restart_lsn         | 0/420A598
(26 rows)
```

Note: If the value of **slot_name** does not exist, the function does not report an error and the return value is **invalid slot name**.

Do not reset a replication slot that is being observed. The error information is as follows:

a. If **slot_name** is empty, the following error is reported: "ERROR: inputString should not be NULL".

b. If **slot_name** is not empty but does not exist, no error is reported but "invalid slot name" is displayed.

c. If **slot_name** is not empty but the replication slot corresponding to **slot_name** is being observed, no error is reported but "can't reset during observing! use gs_logical_decode_stop_observe to stop." is displayed.

- gs_logical_decode_start_observe('slot_name', window, interval)

  Description: Enables logical replication performance sampling.

  Parameter:

  – slot_name

    Indicates the name of the streaming replication slot.

    Value range: a string, supporting only lowercase letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.). One or two periods cannot be used alone as the replication slot name.

  – window

    Specifies the sampling window.

    Value range: an integer ranging from 2 to 1024. Sampling data in the latest *interval* x *window* period is collected.

  – interval

    Specifies the performance monitoring interval, in seconds.

    Value range: interval type. The minimum value is 1s and the maximum value is 1 minute. Sampling data in the latest *interval* x *window* period is collected.

    Return type: text

  Example:

```
gaussdb=# SELECT * FROM gs_logical_decode_start_observe('replication_slot',20,5);
 gs_logical_decode_start_observe
---------------------------------
 OK
```

```
(1 row)

gaussdb=# select * from gs_logical_decode_start_observe('replication_slot',20,5);
 gs_logical_decode_start_observe
---------------------------------
 observe has started!
(1 row)
```

Note: If the value of **slot_name** does not exist, the function does not report an error and the return value is **invalid slot name**.

Do not enable the observe function for a replication slot that is being observed. The error information is as follows:

a. If **slot_name** is empty, the following error is reported: "ERROR: inputString should not be NULL".

b. If **slot_name** is not empty but does not exist, no error is reported but "invalid slot name" is displayed.

c. If the value of **window** is less than **2**, "window has to be >= 2" is displayed.

d. If the value of **window** is greater than **1024**, "window has to be <= 1024" is displayed.

e. If the value of **interval** is less than 1s, "sample interval has to be >= 1s" is displayed.

f. If the value of **interval** is greater than 60s, "sample interval has to be <= 60s" is displayed.

g. If **slot_name** is not empty but the observe function has been enabled for the replication slot corresponding to **slot_name**, no error is reported and the message "observe has started!" is displayed.

- gs_logical_decode_stop_observe('slot_name')

  Description: Stops logical replication performance sampling.

  Parameter:

  – slot_name

    Indicates the name of the streaming replication slot.

    Value range: a string, supporting only lowercase letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.). One or two periods cannot be used alone as the replication slot name.

  – Return type: text

  Example:

```
gaussdb=# SELECT * FROM gs_logical_decode_stop_observe('replication_slot');
 gs_logical_decode_stop_observe
---------------------------------
 OK
(1 row)

gaussdb=# SELECT * FROM gs_logical_decode_stop_observe('replication_slot');
 gs_logical_decode_stop_observe
---------------------------------
 observe not started!
(1 row)
```

Note: If the value of **slot_name** does not exist, the function does not report an error and the return value is **invalid slot name**.

Do not disable the observe function for a replication slot that is not observed. The error information is as follows:

a. If **slot_name** is empty, the following error is reported: "ERROR: inputString should not be NULL".

b. If **slot_name** is not empty but does not exist, no error is reported but "invalid slot name" is displayed.

c. If **slot_name** is not empty but the observe function has been disabled for the replication slot corresponding to **slot_name**, no error is reported and the message "observe not started!" is displayed.

- gs_logical_decode_observe_data('slot_name')

  Description: Displays the original logical replication performance sampling data.

  Parameter:

  – slot_name

    Indicates the name of the streaming replication slot.

    Value range: a string, supporting only lowercase letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.). One or two periods cannot be used alone as the replication slot name.

  – Return type: SETOF record

  Example:

```
gaussdb=# SELECT * FROM gs_logical_decode_observe_data('replication_slot');
 slot_name |          sample_time          | reader_lsn | wal_read_total_time | wal_wait_total_time |
parser_total_time | decoder_total_time | sender_total_time | net_send_total_time |
net_send_total_bytes | transaction_count | big_
transaction_count | sent_transaction_count | spill_transaction_count | spill_disk_bytes
-----------+-------------------------------+------------+---------------------+---------------------
+-------------------+--------------------+-------------------+---------------------+---------------------
+-------------------+-----
-----------------+-----------------------+-----------------------+------------------
 repl      | 2023-01-12 20:09:40.416798+08 |   49447976 |           776846657 |           776846244
|                65 | {46,11,11,6}       |                56 |                   0 |                   0 |
                   0 |                     0 |                   0 |                0
 repl      | 2023-01-12 20:09:45.416849+08 |   49447976 |           776846657 |           776846244
|                65 | {46,11,11,6}       |                56 |                   0 |                   0 |
                   0 |                     0 |                   0 |                0
 repl      | 2023-01-12 20:09:50.417006+08 |   49447976 |           776846657 |           776846244
|                65 | {46,11,11,6}       |                56 |                   0 |                   0 |
                   0 |                     0 |                   0 |                0
 repl      | 2023-01-12 20:09:55.417057+08 |   49447976 |           776846657 |           776846244
|                65 | {46,11,11,6}       |                56 |                   0 |                   0 |
                   0 |                     0 |                   0 |                0
 repl      | 2023-01-12 20:10:00.417115+08 |   49447976 |           776846657 |           776846244
|                65 | {46,11,11,6}       |                56 |                   0 |                   0 |
                   0 |                     0 |                   0 |                0
 repl      | 2023-01-12 20:10:05.417165+08 |   49447976 |           776846657 |           776846244
|                65 | {46,11,11,6}       |                56 |                   0 |                   0 |
                   0 |                     0 |                   0 |                0
 repl      | 2023-01-12 20:10:10.417217+08 |   49447976 |           776846657 |           776846244
|                65 | {46,11,11,6}       |                56 |                   0 |                   0 |
                   0 |                     0 |                   0 |                0
 repl      | 2023-01-12 20:10:15.417271+08 |   49447976 |           776846657 |           776846244
|                65 | {46,11,11,6}       |                56 |                   0 |                   0 |
                   0 |                     0 |                   0 |                0
 repl      | 2023-01-12 20:10:20.417342+08 |   49448264 |           836085882 |           836085442
|                67 | {46,11,11,8}       |                58 |                   0 |                   0 |
                   0 |                     0 |                   0 |                0
 repl      | 2023-01-12 20:10:25.417433+08 |   49448264 |           836085882 |           836085442
|                67 | {46,11,11,8}       |                58 |                   0 |                   0 |
                   0 |                     0 |                   0 |                0
 repl      | 2023-01-12 20:10:30.417526+08 |   49448264 |           836085882 |           836085442
|                67 | {46,11,11,8}       |                58 |                   0 |                   0 |
                   0 |                     0 |                   0 |                0
```



```
 repl      | 2023-01-12 20:10:35.417532+08 |   49448264 |          836085882 |          836085442
|              67 | {46,11,11,8}       |              58 |               0 |                 0 |                0 |
               0 |                  0 |                 0 |              0
 repl      | 2023-01-12 20:10:40.417644+08 |   49448264 |          836085882 |          836085442
|              67 | {46,11,11,8}       |              58 |               0 |                 0 |                0 |
               0 |                  0 |                 0 |              0
 repl      | 2023-01-12 20:10:45.417763+08 |   49448264 |          836085882 |          836085442
|              67 | {46,11,11,8}       |              58 |               0 |                 0 |                0 |
               0 |                  0 |                 0 |              0
(14 rows
```

Note: If the value of **slot_name** does not exist, the function does not report an error and the return value is an empty record.

- gs_logical_decode_observe('slot_name')

  Description: Displays logical replication performance data.

  Parameter:

  – slot_name

    Indicates the name of the streaming replication slot.

    Value range: a string, supporting only lowercase letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.). One or two periods cannot be used alone as the replication slot name.

  – Return type: SETOF record

  Example:

```
gaussdb=# SELECT * FROM gs_logical_decode_observe('replication_slot');
    slot_name     |          sample_time          | logical_decode_rate | wal_read_rate | parser_rate  |
decoder_rate  |  sender_rate  | net_send_rate
------------------+-------------------------------+---------------------+---------------+--------------
+----------------+---------------+---------------
 replication_slot | 2023-01-12 20:16:50.42448+08  | 0.000               | 0.000         | 0.000        |
0.000          | 0.000         | 0.000
 replication_slot | 2023-01-12 20:16:55.424537+08 | 0.000               | 0.000         | 0.000        |
0.000          | 0.000         | 0.000
 replication_slot | 2023-01-12 20:17:00.424641+08 | 0.000               | 0.000         | 0.000        |
0.000          | 0.000         | 0.000
 replication_slot | 2023-01-12 20:17:05.424645+08 | 0.000               | 0.000         | 0.000        |
0.000          | 0.000         | 0.000
 replication_slot | 2023-01-12 20:17:10.424795+08 | 0.000               | 0.000         | 0.000        |
0.000          | 0.000         | 0.000
 replication_slot | 2023-01-12 20:17:15.424848+08 | 0.000               | 0.000         | 0.000        |
0.000          | 0.000         | 0.000
 replication_slot | 2023-01-12 20:17:20.424849+08 | 57.600              | 699029.126    | 96000000.000 |
1152000000.000 | 144000000.000 | 0.000
 replication_slot | 2023-01-12 20:17:25.424959+08 | 0.000               | 699029.126    | 96000000.000 |
1152000000.000 | 144000000.000 | 0.000
 replication_slot | 2023-01-12 20:17:30.42496+08  | 0.000               | 699029.126    | 96000000.000 |
1152000000.000 | 144000000.000 | 0.000
 replication_slot | 2023-01-12 20:17:35.425059+08 | 0.000               | 699029.126    | 96000000.000 |
1152000000.000 | 144000000.000 | 0.000
```

Note: If the value of **slot_name** does not exist, the function does not report an error and the return value is an empty record. If the denominator is 0, the latest collected valid data is returned. If the denominator is not 0 and the numerator is 0, **0** is returned.

Formula:

logical_decode_rate = (reader_lsn1 - reader_lsn2) / (sample_time1 - sample_time2)

wal_read_rate = (reader_lsn1 - reader_lsn2) / (wal_read_total_time1 - wal_read_total_time2) - (wal_wait_total_time1 - wal_wait_total_time2)

parser_rate = (reader_lsn1 - reader_lsn2) / (parser_total_time1 - parser_total_time2)

decoder_rate= (reader_lsn1 - reader_lsn2) / avg(decoder_total_time1[i]-decoder_total_time2[i])

sender_rate = (reader_lsn1 - reader_lsn2) / (sender_total_time1 - sender_total_time2) - (net_send_total_time1 - net_send_total_time2)

sender_rate = (net_sent_bytes1 - net_sent_bytes2) / (net_send_total_time1 - net_send_total_time2) - (net_wait_total_time1 - net_wait_total_time2)

- gs_logical_decode_observe_status('slot_name')

  Description: Queries the monitoring status of a specified logical decoding task.

  Parameter:

  – slot_name

    Indicates the name of the streaming replication slot.

    Value range: a string, supporting only lowercase letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.). One or two periods cannot be used alone as the replication slot name.

  – Return type: text

  Example:

```
gaussdb=# SELECT * FROM gs_logical_decode_observe_status('replication_slot');
 gs_logical_decode_observe_status
----------------------------------
 START
(1 row)

gaussdb=# SELECT * FROM gs_logical_decode_observe_status('replication_slot');
 gs_logical_decode_observe_status
----------------------------------
 invalid slot name
(1 row)

gaussdb=# SELECT * FROM gs_logical_decode_stop_observe('replication_slot');
 gs_logical_decode_stop_observe
--------------------------------
 OK
(1 row)

gaussdb=# SELECT * FROM gs_logical_decode_observe_status('replication_slot');
 gs_logical_decode_observe_status
----------------------------------
 STOP
(1 row)
```

  Note: If the value of **slot_name** does not exist, the function does not report an error and the return value is **invalid slot name**.

- gs_get_parallel_decode_thread_info()

  Description: Executes on the DN where parallel decoding is performed and returns the thread information of parallel decoding.

  Return type: int64, text, text, int

  Example:

```
gaussdb=# SELECT * FROM gs_get_parallel_decode_thread_info();
    thread_id    | slot_name | thread_type | seq_number
-----------------+-----------+-------------+------------
 140335364699904 | slot1     | sender      |          1
 140335214098176 | slot1     | reader      |          1
```



```
140335325312768 | slot1    | decoder    |         1
140335291750144 | slot1    | decoder    |         2
140335274968832 | slot1    | decoder    |         3
140335258187520 | slot1    | decoder    |         4
140335165404928 | slot2    | sender     |         1
140335022864128 | slot2    | reader     |         1
140335129818880 | slot2    | decoder    |         1
140335113037568 | slot2    | decoder    |         2
(10 rows)
```

Note: In the return values, **thread_id** indicates the thread ID, **slot_name** indicates the replication slot name, **thread_type** indicates the thread type (including the sender, reader and decoder), and **seq_number** indicates the sequence number of each thread with same type in the current replication slot. Each parallel decoding connection only has one sender and reader. Therefore, the sequence numbers of the sender and reader are both 1. The sequence numbers of the decoder are arranged from 1 to the decoding degree of parallelism (DOP) of the current replication slot.

- pg_replication_origin_create (node_name)

  Description: Creates a replication source with a given external name and returns the internal ID assigned to it.

  Note: The user who calls this function must have the SYSADMIN permission.

  Parameter:

  – node_name

    Name of the replication source to be created.

    Value range: a string, supporting only letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.).

  Return type: oid

- pg_replication_origin_drop (node_name)

  Description: Deletes a previously created replication source, including any associated replay progress.

  Note: The user who calls this function must have the SYSADMIN permission.

  Parameter:

  – node_name

    Name of the replication source to be deleted.

    Value range: a string, supporting only letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.).

- pg_replication_origin_oid (node_name)

  Description: Searches for a replication source by name and returns the internal ID. If no such replication source is found, an error is thrown.

  Note: The user who calls this function must have the SYSADMIN permission.

  Parameter:

  – node_name

    Specifies the name of the replication source to be queried.

    Value range: a string, supporting only letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.).

  Return type: oid

- pg_replication_origin_session_setup (node_name)

Description: Marks the current session for replaying from a given origin, allowing you to track replay progress. This parameter can be used only when no origin is selected. Run the **pg_replication_origin_session_reset** command to cancel the configuration.

Note: The user who calls this function must have the SYSADMIN permission.

Parameter:

– node_name

Name of the replication source.

Value range: a string, supporting only letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.).

- pg_replication_origin_session_reset ()

  Description: Cancels the **pg_replication_origin_session_setup()** effect.

  Note: The user who calls this function must have the SYSADMIN permission.

- pg_replication_origin_session_is_setup ()

  Description: Returns a true value if a replication source is selected in the current session.

  Note: The user who calls this function must have the SYSADMIN permission.

  Return type: Boolean

- pg_replication_origin_session_progress (flush)

  Description: Returns the replay position of the replication source selected in the current session.

  Note: The user who calls this function must have the SYSADMIN permission.

  Parameter:

  – flush

  Determines whether the corresponding local transaction has been written to disk.

  Value range: Boolean

  Return type: LSN

- pg_replication_origin_xact_setup (origin_lsn, origin_timestamp)

  Description: Marks the current transaction as recommitted at a given LSN and timestamp. This function can be called only when **pg_replication_origin_session_setup** is used to select a replication source.

  Note: The user who calls this function must have the SYSADMIN permission.

  Parameter:

  – origin_lsn

  Position for replaying the replication source.

  Value range: LSN

  – origin_timestamp

  Time point when a transaction is committed.

  Value range: timestamp with time zone

- pg_replication_origin_xact_reset ()

  Description: Cancels the **pg_replication_origin_xact_setup()** effect.

  Note: The user who calls this function must have the SYSADMIN permission.

- pg_replication_origin_advance (node_name, lsn)

  Description:

  Sets the replication progress of a given node to a given position. This is primarily used to set the initial position, or to set a new position after a configuration change or similar change.

  Note: Improper use of this function may cause inconsistent replication data.

  Note: The user who calls this function must have the SYSADMIN permission.

  Parameter:

  – node_name

    Name of an existing replication source.

    Value range: a string, supporting only letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.).

  – lsn

    Position for replaying the replication source.

    Value range: LSN

- pg_replication_origin_progress (node_name, flush)

  Description: Returns the position for replaying the given replication source.

  Note: The user who calls this function must have the SYSADMIN permission.

  Parameter:

  – node_name

    Name of the replication source.

    Value range: a string, supporting only letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.).

  – flush

    Determines whether the corresponding local transaction has been flushed to disk.

    Value range: Boolean

- pg_show_replication_origin_status()

  Description: Displays the replication status of the replication source.

  Note: The user who calls this function must have the SYSADMIN permission.

  Return type:

  – **local_id**: OID, which specifies the ID of the replication source.
  – **external_id**: text, which specifies the name of the replication source.
  – **remote_lsn**: LSN of the replication source.
  – **local_lsn**: local LSN.

- gs_get_distribute_decode_status()

  Description: Obtains the distributed decoding status details (by replication slot) on the current node. This function is not supported in the centralized version, and an error is reported.

  Return type: text, int, int, bigint, xid, xid, text, text, text

- gs_get_distribute_decode_status_detail()

Description: Obtains the distributed decoding status details (by DN) on the current node. This function is not supported in the centralized version, and an error is reported.

Return type: text, int, bigint, int, int, xid

7.6.26.11 Segment-Page Storage Functions

NOTE

- To create a segment-page table on the Astore storage engine, you need to set **segment** to **on** when creating the table. For example:
  ```
  CREATE TABLE t1(id int) WITH (segment=on, storage_type=astore);
  ```

- The values of the fields related to the segment-page storage function are described as follows:
  a. **forknum**: data file fork.
     Value range: [**0**: mainfork; **1**: fsmfork; **2**: vm fork].
  b. **file id**: data file ID.
     Value range: [**1**: metadata file; **2**–**5**: data file].
  c. **blocks**: extent size.
     Value range: [**1**: No.1 file; **8**: No.2 file; **128**: No.3 file; **1024**: No.4 file; **4096**: No.5 file].
  d. **file_block_id**/**head_block_id**/**block_id**: offset page number of the physical page in the data file and other fields that indicate the page number.
     Value range: [0,4294967294]
  e. **page_ type**: page type.
     Value range:
     Meta pages: **file head**/**file_header**: file header; **spc head**/**spc_header**: space header; **map head**/**map_header**: mapping header; **map page**/**map_pages**: mapping page; **reverse pointer page**/**inverse pointer page**/**ip pages**: reverse pointer page; **segment head page**/**segment head**: segment header page; **level1 page**: level-1 page; **data_pages**/**data extent**: data page; **fork head**: fork header.
     Data pages: **heap**, **uheap**, **btree**, and **ubtree**.
     Unknown pages: **unknown(data extent)**: all-zero segment page whose type cannot be determined; **unknown(fsm indexurq)**: fsm or indexurq page.
  f. **contents**: storage content of a data file.
     Value range: **permanent** (permanent), **unlogged** (no log), **temporary** (global temporary), and **temporary2** (local temporary)
- local_space_shrink(tablespacename TEXT, databasename TEXT)
  Description: Shrinks specified physical segment-page space on the current node. Only the currently connected database can be shrank.
  Return value: empty
- gs_space_shrink(tablespace int4, database int4, extent_type int4, forknum int4)
  Description: Works similar to **local_space_shrink**, that is, shrinks specified physical segment-page space. However, the parameters are different. The

input parameters are the OIDs of the tablespace and database, and the value of **extent_type** is an integer ranging from 2 to 5. Note: The value **1** of **extent_type** indicates segment-page metadata. Currently, the physical file that contains the metadata cannot be shrunk. This function is used only by tools. You are advised not to use it directly.

Return value: empty

- gs_stat_remain_segment_info()

  Description: Runs on the primary node to query extents on the local node with residual data due to faults. By default, only initial users, users with the sysadmin permission, and users with the O&M administrator permission in the O&M mode can view the information. Other users can view the information only after being granted with permissions. This function can be executed only on the primary node. Residual extents are classified into two types: segments that are allocated but not used and extents that are allocated but not used. The main difference is that a segment contains multiple extents. During reclamation, all extents in the segment need to be recycled.

  Return type

Name	Description
node_name	Node name.
space_id	Tablespace ID
db_id	Database ID
block_id	Extent ID
type	Extent type. The options are as follows: **ALLOC_SEGMENT**, **DROP_SEGMENT**, and **SHRINK_EXTENT**.

  The values of **type** are described as follows:

  - **ALLOC_SEGMENT**: When a user creates a segment-page table and the segment is just allocated but the transaction of creating a table is not committed, the node is faulty. As a result, the segment is not used after being allocated.
  - **DROP_SEGMENT**: When a user deletes a segment-page table and the transaction is successfully committed, the bit corresponding to the segment page of the table is not reset and a fault, such as power failure, occurs. As a result, the segment is not used or released.
  - **SHRINK_EXTENT**: When a user shrinks a segment-page table and does not release the idle extent, a fault, such as power failure, occurs. As a result, the extent remains and cannot be reused.

    Example:

```
gaussdb=# SELECT * FROM gs_stat_remain_segment_info();

   node_name   | space_id | db_id | block_id |     type
---------------+----------+-------+----------+---------------
    dn_6001    |    16804 | 16803 |     4157 | ALLOC_SEGMENT
(1 row)
```

- gs_free_remain_segment()

  Description: Runs on the primary node to free segments queried by using the gs_stat_remain_segment_info function in the current database. By default, only initial users, users with the sysadmin permission, and users with the O&M administrator permission in the O&M mode can execute the function. Other users can use the function only after being granted with permissions. This function can be executed only on the primary node.

  Return type: Boolean

- gs_local_stat_remain_segment_info()

  Description: Runs on the primary node to query the residual segment page information of the current node. For details about user permissions and return values, see gs_stat_remain_segment_info.

- gs_local_free_remain_segment()

  Description: Runs on the primary node to clear the residual segment page information of the current database on the current node. For details about user permissions and return values, see gs_free_remain_segment.

- gs_seg_dump_page(tablespace_name name, file_id int4, bucketnode int4, file_block_id bigint, forknum int4 default 0)

  Description: Parses a specified page in segment-page mode and returns the parsed content. Only users with the sysadmin attribute and users with the O&M administrator attribute in O&M mode can execute this function. This function is used to parse physical pages. The parsing result of one page is returned each time. The returned result does not contain the actual user data. This function does not require users to enter the page type. During implementation, the system attempts to determine the page type. If the page type cannot be determined, the system outputs possible parsing results.

  Parameter description

Name	Type	Description
table_space_name	NAME	Tablespace to which a segment belongs. Value range: a valid tablespace name.
file_id	INTEGER	Data file ID. Value range: an int4 value in the range [1,5].
bucketnode	INTEGER	• **0** to **1023** indicate the bucket nodes of a hash bucket table. • **1024** indicates the bucket node of a segment-page ordinary table. • **1025** indicates the bucket node of a segment-page global temporary table. • **1026** indicates the bucket node of a segment-page unlogged table. • **1027** indicates the bucket node of a segment-page local temporary table.
file_block_id	BIGINT	Offset page number of the physical page in the data file. Value range: [0,4294967294]

Name	Type	Description
forknum	INTEGER DEFAULT 0	Fork number of a data file. The default value is **0**. Value range: ● **0**: main fork. ● **1**: FSM fork. ● **2**: VM fork.

Return type

Name	Type	Description
page_type	TEXT	Page type. Value range: ● Service pages: **heap**, **uheap**, **btree**, **ubtree**, **gsivfflat_index**, and **gsdiskann_index** ● Segment-page metapages: **bucket main head**, **file head**, **spc head**, **map head**, **map page**, **reverse pointer page**, **segment head page** and **level1 page**. ● Unknown pages: – **unknown(data extent)**: all-zero segment page whose type cannot be determined. – **unknown(fsm indexurq)**: fsm or indexurq page.
result	TEXT	Parsing result.

For example, perform the following operations after creating a segment-paged ordinary table:

```
gaussdb=# SELECT * FROM gs_seg_dump_page('pg_default', 1, 1024, 4157);
    page_type    |                                  result
-----------------+-----------------------------------------------------------------------------
segment head page | Page information of block 4157/4157                                        +
                  |       pd_lsn: 0/2C90608 ,len 8 ,offset 0                                   +
                  |                                                                            +
                  |       pd_checksum: 0x8A7F, verify success,len 2, offset 8                  +
                  |       pd_flags:                                                            +
                  |       pd_lower: 24, empty, len 2, offset 12                                +
                  |       pd_upper: 8192, old, len 2, offset 14                                +
                  |       pd_special: 8192, size 0, len 2, offset 16                           +
                  |       Page size & version: 8192, 8, len 2, offset 18                       +
                  |       pd_xid_base: 0, len 8, offset 24 pd_multi_base: 0, len 8, offset 32+
                  |       pd_prune_xid: 0, len 4 ,offset 20                                    +
                  |                                                                            +
                  |       Segment head information on this page                                +
                  |             magic: 44414548544E454D                                        +
                  |             lsn is: 0/2C90540                                              +
                  |             nblocks: 0                                                     +
                  |             total_blocks: 8                                                +
                  |             reserved: 0                                                    +
```

```
|              Level 0 slots information on this page                +
|                   The BlockNumber of level0 slots 0 is: 4157        +
|              fork head information on this page                    +
|                   4157(valid)                                  +
|                   4294967295(invalid)                               +
|                   4294967295(invalid)
(1 row)
```

- gs_seg_dump_page(relid oid, bucketnode int, block_id bigint, partition bool default false, forknum int4 default 0)

  Description: Parses a specified page in segment-page mode and returns the parsed content. This function is executed on DNs. Only users with the sysadmin attribute and users with the O&M administrator attribute in O&M mode can execute this function. This function is used to parse logical pages. The parsing result of one page is returned each time. The returned result does not contain the actual user data. This function does not require users to enter the page type. During implementation, the system attempts to determine the page type. If the page type cannot be determined, the system outputs possible parsing results.

  Parameter description

Name	Type	Description
relid	OID	Segment object ID. Value range: a valid segment-page object ID. Otherwise, an error is reported.
bucketnode	INTEGER	• **0** to **1023** indicate the bucket nodes of a hash bucket table. • **1024** indicates the bucket node of a segment-page ordinary table. • **1025** indicates the bucket node of a segment-page global temporary table. • **1026** indicates the bucket node of a segment-page unlogged table. • **1027** indicates the bucket node of a segment-page local temporary table.
block_id	BIGINT	Logical page number. Value range: [0,4294967294]
partition	BOOLEAN DEFAULT FALSE	Specifies whether the segment object is a partition. The default value is **false**. Value range: bool, indicating whether the object corresponding to the input OID is a partition.
forknum	INTEGER DEFAULT 0	Fork number of a data file. The default value is **0**. Value range: • **0**: main fork. • **1**: FSM fork. • **2**: VM fork.

  Return value description

Name	Type	Description
page_type	TEXT	Page type. Value range: • Service pages: **heap**, **uheap**, **btree**, **ubtree**, **gsivfflat_index**, and **gsdiskann_index** • Segment-page metapages: **bucket main head**, **file head**, **spc head**, **map head**, **map page**, **reverse pointer page**, **segment head page** and **level1 page**. • Unknown pages: – **unknown(data extent)**: all-zero segment page whose type cannot be determined. – **unknown(fsm indexurq)**: fsm or indexurq page.
result	TEXT	Parsing result.

In the following example, **relid** must be a valid OID of a multipart page object, and the object must have data so that the page information can be queried:

```
gaussdb=# SELECT * FROM gs_seg_dump_page(16788, 1024, 0);
 page_type |                                                 result
-----------
+-------------------------------------------------------------------------------------------------------------------
----
 heap      | Page information of block
6021/6021                                                                                                        +
           |        pd_lsn: 0/4463418 ,len 8 ,offset 0                                                                 +
           |                                                                                                       +
           |        pd_checksum: 0xD4CD, verify success,len 2, offset
8                                                                                                                +
           |        pd_flags:                                                                                        +
           |        pd_lower: 44, non-empty, len 2, offset
12                                                                                                               +
           |        pd_upper: 8160, old, len 2, offset 14                                                             +
           |        pd_special: 8192, size 0, len 2, offset 16                                                        +
           |        Page size & version: 8192, 6, len 2, offset 18
+
           |        pd_xid_base: 17049, len 8, offset 24 pd_multi_base: 0, len 8, offset
32                                                                                                               +
           |        pd_prune_xid: 17049, len 4 ,offset 20                                                              +
           |                                                                                                       +
           |        Heap tuple information on this page
+
           |                                                                                                       +
           |               Tuple #1 is normal: length 28, offset
8160                                                                                                             +
           |                       (uint64)xmin/xmax/t_cid:
17052/0/0                                                                                                        +
           |                       (uint32)t_xmin/t_xmax: 3/3(check ilm flag to indicate whether t_xmin/
t_xmax is xid or ilm time)+
           |                       ctid:(block 0/0, offset 1)                                                        +
           |                       t_infomask: HEAP_XMAX_INVALID
HEAP_HAS_NO_UID                                                                                                  +
           |                       t_infomask2: Attrs Num: 1                                                           +
           |                       t_hoff: 24                                                                        +
           |                       t_bits: NNNNNNNN                                                                    +
           |        Summary (1 total):  1 normal, 0 unused, 0
dead                                                                                                             +
```

```
        |                                                                                +
        | Normal Heap Page, special space is 0
+
        |                                                                                +
        |
(1 row)
```

- gs_seg_get_spc_location(tablespace_name NAME, bucketnode INTEGER, head_block_id BIGINT, block_id BIGINT)

  Description: Calculates the physical location given a segment and logical page number. Only an administrator can query the information.

  Parameter description

Name	Type	Description
tablespace_name	NAME	Tablespace to which a segment belongs.
bucketnode	INTEGER	• **0** to **1023** indicate the bucket nodes of a hash bucket table. • **1024** indicates the bucket node of a segment-page ordinary table. • **1025** indicates the bucket node of a segment-page global temporary table. • **1026** indicates the bucket node of a segment-page unlogged table. • **1027** indicates the bucket node of a segment-page local temporary table.
head_block_id	BIGINT	Page number of a segment header.
block_id	BIGINT	Logical page number.

  Return value description

Name	Type	Description
extent_id	INTEGER	Logical extent number of the logical page.
extent_size	INTEGER	Size of the logical extent where the logical page is located.
file_id	INTEGER	ID of the data file where the physical page is located.
file_block_id	BIGINT	Offset page number of the physical page in the data file.

  For example, perform the following operations after creating a page-based ordinary table in the tablespace and inserting data:

```
gaussdb=# SELECT * FROM gs_seg_get_spc_location('pg_default', 1024, 4157, 0);
 extent_id | extent_size | file_id | file_block_id
```



```
-----------+-------------+---------+---------------
         0 |           8 |       2 |          4157
(1 row)
```

- gs_seg_get_spc_location(relid OID, bucketnode INTEGER, block_id BIGINT, partition BOOLEAD DEFAULT FALSE, forknum INTEGER DEFAULT 0)

  Description: Calculates the physical location given a segment and logical page number. Only an administrator can query the information.

  Parameter description

Name	Type	Description
relid	OID	Segment object ID.
bucketnode	INTEGER	• **0** to **1023** indicate the bucket nodes of a hash bucket table. • **1024** indicates the bucket node of a segment-page ordinary table. • **1025** indicates the bucket node of a segment-page global temporary table. • **1026** indicates the bucket node of a segment-page unlogged table. • **1027** indicates the bucket node of a segment-page local temporary table.
block_id	BIGINT	Logical page number.
partition	BOOLEAD DEFAULT FALSE	Specifies whether the segment object is a partition. The default value is **FALSE**.
forknum	INTEGER DEFAULT 0	Fork of a segment object. The default value is **0**. Value range: • **0**: main fork. • **1**: FSM fork. • **2**: VM fork.

  Return value description

Name	Type	Description
extent_id	INTEGER	Logical extent number of the logical page.
extent_size	INTEGER	Size of the logical extent where the logical page is located.
file_id	INTEGER	ID of the data file where the physical page is located.
file_block_id	BIGINT	Offset page number of the physical page in the data file.

In the following example, **relid** must be a valid OID of a multipart page object, and the object must have data:

```
gaussdb=# SELECT * FROM gs_seg_get_spc_location(24578,1024,0);
 extent_id | extent_size | file_id | file_block_id
-----------+-------------+---------+---------------
         0 |           8 |       2 |          4157
(1 row)
```

- gs_seg_get_location(block_id BIGINT, as_extent BOOLEAN DEFAULT FALSE)

  Description: Calculates the physical location given a segment and logical page number. Only an administrator can query the information.

  Parameter description

Name	Type	Description
block_id	BIGINT	Logical page number.
as_extent	BOOLEAN DEFAULT FALSE	Specifies whether the **block_id** parameter is an extent number. The default value is **FALSE**.

  Return value description

Name	Type	Description
extent_id	BIGINT	Extent number.
extent_size	INTEGER	Extent size.
extent_offset	INTEGER	Extent offset block number.
level0_slots_idx	INTEGER	Index of the extent in the level-0 slot array in the segment header.
level1_slots_idx	INTEGER	Index of the extent in the level-1 slot array in the segment header.
level1_page_offset	INTEGER	Offset of the extent on the level-1 slot page in the segment header.
segment_blocks	BIGINT	Number of pages containing this page or extended segments.
relative_fno	INTEGER	Number of the relative file that contains this page or extent.

  Example:

```
gaussdb=# SELECT * FROM gs_seg_get_location(4157);
 extent_id | extent_size | extent_offset | level0_slots_idx | level1_slots_idx | level1_page_offset |
segment_blocks | relative_fno
-----------+-------------+---------------+------------------+------------------+--------------------
+----------------+--------------
        47 |         128 |            61 |               47 |                  |                  |           4158 |            3
(1 row)
```


- gs_seg_get_segment_layout()

  Description: Outputs the static segment layout. Only an administrator can query the information.

  Return value description

Name	Type	Description
version	TEXT	Segment-page version. The default value is **1.0**.
section_id	INTEGER	ID of the data section divided by a segment.
section_type	TEXT	Extent type of the segment data section. Value range: ● **meta**: segment header. ● **data**: data.
extent_size	INTEGER	Extent size. The unit is byte.
extent_page_count	INTEGER	Number of extent pages.
extent_count_start	BIGINT	Start extent number.
extent_count_end	BIGINT	End extent number.
total_size	BIGINT	Size of the segment data section. The unit is byte.

  Example:

```
gaussdb=# SELECT * FROM gs_seg_get_segment_layout();
 version | section_id | section_type | extent_size | extent_page_count | extent_count_start |
extent_count_end |   total_size
---------+------------+--------------+-------------+-------------------+--------------------+------------------
+----------------
 1.0     |          1 | meta         |        8192 |                 1 |                  0 |                0 |          8192
 1.0     |          2 | data         |       65536 |                 8 |                  1 |               16 |       1048576
 1.0     |          3 | data         |     1048576 |               128 |                 17 |              143 |     134217728
 1.0     |          4 | data         |     8388608 |              1024 |                144 |              255 |
1073741824
 1.0     |          5 | data         |    33554432 |              4096 |                256 |          1025255 |
34394366541824
(5 rows)
```

- gs_seg_get_datafile_layout()

  Description: Queries the static layout of data files 1 to 5. Only an administrator can query the information.

  Return value description

Name	Type	Description
version	TEXT	Segment-page version. The default value is **1.0**.

Name	Type	Description
seg_storage_type	TEXT	• **segment** indicates common segment-page data. • **hashbucket** indicates hash bucket data.
file_id	INTEGER	Data file ID.
section_id	INTEGER	Data section ID of a data file.
section_type	TEXT	Type of the data file section. Value range: • **file_header** indicates the file header. • **spc_header** indicates the space header. • **map_header** indicates the mapping header. • **map_pages** indicates the mapping page. • **ip_pages(inverse pointer pages)** indicates the reverse pointer page. • **data_pages** indicates the data page.
page_start	BIGINT	Start page number of the data section.
page_end	BIGINT	End page number of the data section.
page_count	BIGINT	Number of pages in the data section.
total_size	BIGINT	Size of the data section. The unit is byte.

Example:

```
gaussdb=# SELECT * FROM gs_seg_get_datafile_layout();

... (There is a large amount of data. Only part of the data is displayed.)
 version | seg_storage_type | file_id | section_id | section_type |  page_start  |   page_end   |
page_count  |   total_size
---------+------------------+---------+------------+--------------+--------------+--------------+-------------
+-----------------
 1.0     | segment          |       1 |          0 | file_header  |            0 |            0 |            1 |           8192
 1.0     | segment          |       1 |          1 | spc_header   |            1 |            1 |            1 |           8192
 1.0     | segment          |       1 |          2 | map_header   |            2 |            2 |            1 |           8192
 1.0     | segment          |       1 |          3 | map_pages    |            3 |           66 |           64 |         524288
 1.0     | segment          |       1 |          4 | ip_pages     |           67 |         4156 |         4090 |
33505280
 1.0     | segment          |       1 |          5 | data_pages   |         4157 |      4147260 |      4143104 |
33940307968
```

- gs_seg_get_slice_layout(file_id INTEGER,bucketnode INTEGER, slice_id INTEGER)

  Description: Outputs the static layout of a given data file fragment. Only an administrator can query the information.

  Parameter description

Name	Type	Description
file_id	INTEGER	Data file ID.
bucketnode	INTEGER	• **0** to **1023** indicate the bucket nodes of a hash bucket table. • **1024** indicates the bucket node of a segment-page ordinary table. • **1025** indicates the bucket node of a segment-page global temporary table. • **1026** indicates the bucket node of a segment-page unlogged table. • **1027** indicates the bucket node of a segment-page local temporary table.
slice_id	INTEGER	Slice file ID.

Return value description

Name	Type	Description
version	TEXT	Segment-page version. The default value is **1.0**.
section_id	INTEGER	Data section ID of a data file.
section_type	INTEGER	Type of the data file section. Value range: • **file_header** indicates the file header. • **spc_header** indicates the space header. • **map_header** indicates the mapping header. • **map_pages** indicates the mapping page. • **ip_pages(inverse pointer pages)** indicates the reverse pointer page. • **data_pages** indicates the data page.
page_start	BIGINT	Start page number of the data section.
page_end	BIGINT	End page number of the data section.
page_count	BIGINT	Number of pages in the data section.
total_size	BIGINT	Size of the data section. The unit is byte.

Example:

```
gaussdb=# SELECT * FROM gs_seg_get_slice_layout(1,1024, 0);
 version | section_id | section_type | page_start | page_end | page_count | total_size
```



```
---------+------------+--------------+------------+----------+------------+------------
 1.0     |          0 | file_header  |          0 |        0 |          1 |       8192
 1.0     |          1 | spc_header   |          1 |        1 |          1 |       8192
 1.0     |          2 | map_header   |          2 |        2 |          1 |       8192
 1.0     |          3 | map_pages    |          3 |       66 |         64 |     524288
 1.0     |          4 | ip_pages     |         67 |     4156 |       4090 |   33505280
 1.0     |          5 | data_pages   |       4157 |   131071 |     126915 | 1039687680
(6 rows)
```

- gs_seg_get_segment(tablespace_name NAME, bucketnode INTEGER, head_block_id BIGINT)

  Description: Outputs the segment header information of the segment header page in the segment header file of the tablespace. Only an administrator can query the information.

  Parameter description

Name	Type	Description
tablespace_name	NAME	Tablespace to which a segment belongs.
bucketnode	INTEGER	• **0** to **1023** indicate the bucket nodes of a hash bucket table. • **1024** indicates the bucket node of a segment-page ordinary table. • **1025** indicates the bucket node of a segment-page global temporary table. • **1026** indicates the bucket node of a segment-page unlogged table. • **1027** indicates the bucket node of a segment-page local temporary table.
head_block_id	BIGINT	Page number of the segment header.

  Return value description

Name	Type	Description
blocks	BIGINT	Number of logical pages of a segment.
total_blocks	BIGINT	Number of physical pages of a segment.
extents	INTEGER	Number of logical extents of a segment.
total_extents	INTEGER	Number of physical extents of a segment.
head_lsn	TEXT	Segment header LSN.
level0_slots	BIGINT[]	Level-0 slot array of segment extent mapping.
level1_slots	BIGINT[]	Level-1 slot array of segment extent mapping.

fork_head	BIGINT[]	Fork head array of a segment.

For example, perform the following operations after creating a segment-paged ordinary table:

```
gaussdb=# SELECT * FROM gs_seg_get_segment('pg_default', 1024, 4157);
 blocks | total_blocks | extents | total_extents | head_lsn | level0_slots | level1_slots |   fork_head
--------+--------------+---------+---------------+----------+--------------+--------------+---------------
      9 |           16 |       2 |             2 | 62211744 | {4157,4165}  | {}           | {4157,4158,-1}
(1 row)
```

- gs_seg_get_segment(relid OID, bucketnode INTEGER, partition BOOLEAD DEFAULT FALSE, forknum INTEGER DEFAULT 0)

  Description: Outputs the corresponding segment header information based on **relid** and **bucketnode**. Only an administrator can query the information.

  Parameter description

Name	Type	Description
relid	OID	Table OID.
bucketnode	INTEGER	• **0** to **1023** indicate the bucket nodes of a hash bucket table. • **1024** indicates the bucket node of a segment-page ordinary table. • **1025** indicates the bucket node of a segment-page global temporary table. • **1026** indicates the bucket node of a segment-page unlogged table. • **1027** indicates the bucket node of a segment-page local temporary table.
partition	BOOLEAN DEFAULT FALSE	Specifies whether the segment object is a partition. The default value is **false**. Value range: bool, indicating whether the object corresponding to the input OID is a partition.
forknum	INTEGER DEFAULT 0	Fork number of a data file. The default value is **0**. Value range: • **0**: main fork. • **1**: FSM fork. • **2**: VM fork.

  Return value description

Name	Type	Description
blocks	BIGINT	Number of logical pages of a segment.

Name	Type	Description
total_blocks	BIGINT	Number of physical pages of a segment.
extents	INTEGER	Number of logical extents of a segment.
total_extents	INTEGER	Number of physical extents of a segment.
head_lsn	TEXT	Segment header LSN.
level0_slots	BIGINT[]	Level-0 slot array of segment extent mapping.
level1_slots	BIGINT[]	Level-1 slot array of segment extent mapping.
fork_head	BIGINT[]	Fork head array of a segment.

In the following example, **relid** must be a valid OID of a multipart page object:

```
gaussdb=# SELECT * FROM gs_seg_get_segment(16768, 1024);
 blocks | total_blocks | extents | total_extents | head_lsn | level0_slots | level1_slots |    fork_head
--------+--------------+---------+---------------+----------+--------------+--------------+-----------------
      9 |           16 |       2 |             2 | 62211744 | {4157,4165}  | {}           | {4157,4158,4294967295}
(1 row)
```

- gs_seg_get_extents(tablespace_name NAME, bucketnode INTEGER, head_block_id BIGINT)

  Description: Outputs all extents of the segment object on the segment header page in the segment header file of the tablespace, including **segment head**, **fork head**, **level1 page** in file 1 and data extents in files 2 to 5. Only an administrator can query the information.

  Parameter description

Name	Type	Description
tablespace_name	NAME	Tablespace to which a segment belongs.
bucketnode	INTEGER	• **0** to **1023** indicate the bucket nodes of a hash bucket table. • **1024** indicates the bucket node of a segment-page ordinary table. • **1025** indicates the bucket node of a segment-page global temporary table. • **1026** indicates the bucket node of a segment-page unlogged table. • **1027** indicates the bucket node of a segment-page local temporary table.
head_block_id	BIGINT	Page number of the segment header.

Return value description

Name	Type	Description
extent_id	INTEGER	Logical extent number.
file_id	INTEGER	ID of the data file where the extent is located.
forknum	INTEGER	Segment fork. Value range: ● **0**: main fork. ● **1**: FSM fork. ● **2**: VM fork.
block_id	BIGINT	Start page number in the data file where the extent is located.
blocks	INTEGER	Extent size. Value range: ● **1** indicates file 1. ● **8** indicates file 2. ● **128** indicates file 3. ● **1024** indicates file 4. ● **4096** indicates file 5.
usage_type	TEXT	Usage type of an extent. Value range: ● **segment head** indicates the segment header. ● **fork head** indicates the fork header. ● **level1 page** indicates the level-1 page. ● **data extent** indicates data extents.

For example, perform the following operations after creating a segment-page ordinary table in the tablespace:

```
gaussdb=# SELECT * FROM gs_seg_get_extents('pg_default', 1024, 4157);
 extent_id | file_id | forknum | block_id | blocks |  usage_type
-----------+---------+---------+----------+--------+--------------
           |       1 |       0 |     4157 |      1 | segment head
         0 |       2 |       0 |     4157 |      8 | data extent
         1 |       2 |       0 |     4165 |      8 | data extent
(3 rows)
```

- gs_seg_get_extents(relid OID, bucketnode INTEGER, partition BOOLEAD DEFAULT FALSE, forknum INTEGER DEFAULT 0)

  Description: Outputs all extents of the segment object on the segment header page in the corresponding segment header file, including **segment head**, **fork head**, **level1 page** in file 1 and data extents in files 2 to 5. Only an administrator can query the information.

  Parameter description

Name	Type	Description
relid	OID	Table OID.
bucketnode	INTEGER	• **0** to **1023** indicate the bucket nodes of a hash bucket table. • **1024** indicates the bucket node of a segment-page ordinary table. • **1025** indicates the bucket node of a segment-page global temporary table. • **1026** indicates the bucket node of a segment-page unlogged table. • **1027** indicates the bucket node of a segment-page local temporary table.
partition	BOOLEAN DEFAULT FALSE	Specifies whether the segment object is a partition. The default value is **false**. Value range: bool, indicating whether the object corresponding to the input OID is a partition.
forknum	INTEGER DEFAULT 0	Fork number of a data file. The default value is **0**. Value range: • **0**: main fork. • **1**: FSM fork. • **2**: VM fork.

Return value description

Name	Type	Description
extent_id	INTEGER	Logical extent number.
file_id	INTEGER	ID of the data file where the extent is located.
forknum	INTEGER	Segment fork. Value range: • **0**: main fork. • **1**: FSM fork. • **2**: VM fork.
block_id	BIGINT	Start page number in the data file where the extent is located.

Name	Type	Description
blocks	INTEGER	Extent size. Value range: • **1** indicates file 1. • **8** indicates file 2. • **128** indicates file 3. • **1024** indicates file 4. • **4096** indicates file 5.
usage_type	TEXT	Usage type of an extent. Value range: • **segment head** indicates the segment header. • **fork head** indicates the fork header. • **level1 page** indicates the level-1 page. • **data extent** indicates data extents.

In the following example, **relid** must be a valid OID of a multipart page object:

```
gaussdb=# SELECT * FROM gs_seg_get_extents(16768, 1024);
 extent_id | file_id | forknum | block_id | blocks |  usage_type
-----------+---------+---------+----------+--------+--------------
           |       1 |       0 |     4157 |      1 | segment head
         0 |       2 |       0 |     4157 |      8 | data extent
         1 |       2 |       0 |     4165 |      8 | data extent
(3 rows)
```

- gs_seg_free_spc_remain_segment(tablespace_name NAME, head_file_id INTEGER, bucketnode INTEGER, head_block_id BIGINT)

  Description: Releases the page occupied by the segment page-based residual segment in file 1 in a specified tablespace. You can query residual segments in the GS_SEG_SPC_REMAIN_SEGMENTS view. Only an administrator can query the information. This query can be executed only on the primary node.

NOTICE

- This function has a defect and is planned to be reconstructed in later versions to completely solve the residual segment-page problem. This function is an offline segment-page clearing method.
- To ensure the consistency between the queried residual segments and residual extents and the consistency of residual clearing, this function and the GS_SEG_SPC_REMAIN_SEGMENTS view must be executed in the DDL-\DML-restricted state. The current version does not provide the DML/DDL restriction capability. Therefore, when using this feature, ensure that the operating environment is in the DDL-\DML-restricted state.
- This function can be used only when **enable_segment_remain_cleanup** is set to **off**. For details about how to enable it, see "Configuring Running Parameters > GUC Parameters > Developer Options" in *Administrator Guide*.

Parameter description

Name	Type	Description
tablespace_name	NAME	Tablespace name.
head_file_id	INTEGER	ID of a data file in which a segment header of a segment-page-type residual segment is located. Value range: **1**
bucketnode	INTEGER	• **0** to **1023** indicate the bucket nodes of a hash bucket table. • **1024** indicates the bucket node of a segment-page ordinary table. • **1025** indicates the bucket node of a segment-page global temporary table. • **1026** indicates the bucket node of a segment-page unlogged table. • **1027** indicates the bucket node of a segment-page local temporary table.
head_block_id	BIGINT	Page number of a segment header.

Return type: void

Example:

```
gaussdb=# SELECT * FROM gs_seg_free_spc_remain_segment('pg_default', 1, 1024, 4159);
 gs_seg_free_spc_remain_segment
--------------------------------

(1 row)
```

- gs_seg_free_spc_remain_extent(tablespace_name NAME, file_id INTEGER, bucketnode INTEGER, forknum INTEGER, block_id BIGINT)

  Description: Releases residual segment-page-type isolated extents in a specified tablespace. Residual isolated extents can be queried in the

GS_SEG_SPC_REMAIN_EXTENTS view. This function can also clear the segment header of the segment queried in the GS_SEG_SPC_REMAIN_SEGMENTS view. However, this action will expose other extents (such as **fork head**, **level1 page**, and **data extent**) in the segment where the cleared segment header is located in the GS_SEG_SPC_REMAIN_EXTENTS view. Only an administrator can query the information. This query can be executed only on the primary node.

NOTICE

- This function has a defect and is planned to be reconstructed in later versions to completely solve the residual segment-page problem. This function is an offline segment-page clearing method.
- To ensure the consistency between the queried residual segments and residual extents and the consistency of residual clearing, this function and the GS_SEG_SPC_REMAIN_EXTENTS view must be executed in the DDL- or DML-restricted state. The current version does not provide the DML/DDL restriction capability. Therefore, when using this feature, ensure that the operating environment is in the DDL-\DML-restricted state.
- This function can be used only when **enable_segment_remain_cleanup** is set to **off**. For details about how to enable it, see "Configuring Running Parameters > GUC Parameters > Developer Options" in *Administrator Guide*.

Parameter description

Name	Type	Description
tablespace_name	NAME	Tablespace name.
file_id	INTEGER	ID of the data file where the segment-page-type residual extent is located. Value range: files 1 to 5.
bucketnode	INTEGER	• **0** to **1023** indicate the bucket nodes of a hash bucket table. • **1024** indicates the bucket node of a segment-page ordinary table. • **1025** indicates the bucket node of a segment-page global temporary table. • **1026** indicates the bucket node of a segment-page unlogged table. • **1027** indicates the bucket node of a segment-page local temporary table.

Name	Type	Description
forknum	INTEGER	Fork number of a data file. Value range: • **0**: main fork. • **1**: FSM fork. • **2**: VM fork.
block_id	BIGINT	Page number of a segment header.

Return type: void

Example:

```
gaussdb=# SELECT * FROM gs_seg_free_spc_remain_extent('pg_default', 1, 1024, 0, 4159);
 gs_seg_free_spc_remain_extent
-------------------------------

(1 row)
```

- gs_seg_get_datafiles(database_name NAME)

  Description: Displays information about all data files of an instance. Only an administrator can query the information.

  Parameter description

Name	Type	Description
database_name	NAME	Database name. The default value is **current_database()**, indicating the name of the current database.

  Return value description

Name	Type	Description
file_name	TEXT	Data file name, for example, **base/17467/2_fsm**.
file_id	INTEGER	Data file ID. Value range: files 1 to 5.
bucketnode	INTEGER	• **0** to **1023** indicate the bucket nodes of a hash bucket table. • **1024** indicates the bucket node of a segment-page ordinary table. • **1025** indicates the bucket node of a segment-page global temporary table. • **1026** indicates the bucket node of a segment-page unlogged table. • **1027** indicates the bucket node of a segment-page local temporary table.

Name	Type	Description
forknum	INTEGER	Fork type of a data file.
tablespace_name	NAME	Name of the tablespace to which a data file belongs.
contents	TEXT	Storage content of a data file. Value range: • **permanent**: permanent. • **unlogged**: no log. • **temporary**: global temporary • **temporary2**: local temporary
extent_size	INTEGER	Extent size of a data file.
meta_blocks	BIGINT	Number of allocated metadata pages of a data file.
data_blocks	BIGINT	Number of allocated data pages of a data file.
total_blocks	BIGINT	Total number of physical pages in a data file.
high_water_mark	BIGINT	High watermark of the number of pages used by a data file.
utilization	REAL	Percentage of used blocks to the total number of blocks, that is, (**data_blocks** + **meta_blocks**)/**total_blocks**.

For example, perform the following operations after creating a segment-paged table:

```
gaussdb=# SELECT * FROM gs_seg_get_datafiles();
  file_name   | file_id | bucketnode | forknum | tablespace_name | contents  | extent_size | meta_blocks
| data_blocks | total_blocks | high_water_mark | utilization
--------------+---------+------------+---------+-----------------+-----------+-------------+-------------
+-------------+--------------+-----------------+-------------
 base/15949/1 |       1 |       1024 |       0 | pg_default      | permanent |           1 |        4157 |           1
|        16384 |      1048702976 | 2.24208e-44
 base/15949/2 |       2 |       1024 |       0 | pg_default      | permanent |           8 |        4157 |           8
|        16384 |      1048717312 | 2.24208e-44
(2 rows)
```

- gs_seg_get_spc_extents(tablespace_name NAME, file_id INTEGER, bucketnode INTEGER, forknum INTEGER, skip_unused BOOLEAN DEFAULT TRUE)

  Description: Obtains all extent information of a specified tablespace. Only an administrator can query the information.

  Parameter description

Name	Type	Description
tablespace_name	NAME	Tablespace name.
file_id	INTEGER	Data file ID.
bucketnode	INTEGER	• **0** to **1023** indicate the bucket nodes of a hash bucket table. • **1024** indicates the bucket node of a segment-page ordinary table. • **1025** indicates the bucket node of a segment-page global temporary table. • **1026** indicates the bucket node of a segment-page unlogged table. • **1027** indicates the bucket node of a segment-page local temporary table.
forknum	INTEGER	Data file fork. Value range: • **0**: main fork. • **1**: FSM fork. • **2**: VM fork.
skip_unused	BOOLEAN DEFAULT TRUE	Specifies whether to output only allocated extents. The default value is **TRUE**, indicating that only allocated extents are output.

Return value description

Name	Type	Description
block_id	BIGINT	Start page number of a data extent.
blocks	INTEGER	Data extent size. Value range: • **1** indicates file 1. • **8** indicates file 2. • **128** indicates file 3. • **1024** indicates file 4. • **4096** indicates file 5.

Name	Type	Description
contents	TEXT	Stores content. Value range: ● **permanent**: permanent. ● **unlogged**: no log. ● **temporary**: global temporary. ● **temporary2**: local temporary.
in_used	TEXT	Specifies whether a resource has been allocated. Value range: ● **Y**: allocated. ● **N**: not allocated.
mapblock_location	TEXT	Position of the extent in the map block. Format: (page_id, offset).
head_file_id	INTEGER	ID of the data file where the segment header is located.
head_block_id	BIGINT	Page number of the segment header.
usage_type	TEXT	Usage type of an extent. Value range: ● **segment head** indicates the segment header. ● **fork head** indicates the fork header. ● **level1 page** indicates the level-1 page. ● **data extent** indicates data extents.
remain_flag	TEXT	Specifies whether it is a residual extent after the SHRINK operation. Value range: ● **Y**: residual extent upon SHRINK. ● **N**: not a residual extent upon SHRINK.
special_data	INTEGER	Special data section of the reverse pointer corresponding to an extent.
ipblock_location	TEXT	Position of the reverse pointer corresponding to an extent. Format: (block_id, offset).

Example:

```
gaussdb=# SELECT * FROM gs_seg_get_spc_extents('pg_default', 1,1024, 0);
 block_id | blocks | contents  | in_used | mapblock_location | head_file_id | head_block_id |  usage_type
| remain_flag | special_data | ipblock_location
----------+--------+-----------+---------+-------------------+--------------+---------------+--------------
+-------------+--------------+------------------
    4157 |      1 | permanent | Y       | (4157,0)          |            1 |          4157 | segment head | N
|            0 | (67,0)
```

```
    4158 |      1 | permanent | Y       | (4157,1)          |           1 |          4158 | fork head    | N
|           1 | (67,1)
(2 rows)
```

7.6.26.12 Hash Bucket System Functions

- gs_redis_get_plan(origin_group_id OID, target_group_id OID)

  Description: Obtains a complete migration plan. This function is not supported in the current version.

- gs_redis_get_bucket_statistics

  Description: Obtains the log flow transmission status. This function is not supported in the current version.

- gs_redis_set_distributed_db(db_name CSTRING)

  Description: Sets the database that is being redistributed using hash bucket. This function is not supported in the current version.

- gs_redis_hashbucket_update_segment_header(origin_group_id OID, target_group_id OID)

  Description: Updates the headers of all hash bucket tables in the current database. This function is not supported in the current version.

- gs_redis_local_get_segment_header(table_name REGCLASS, bucketlist OIDVECTOR_EXTEND)

  Description: Obtains the header of a segment-page table. This function is not supported in the current version.

- gs_redis_local_update_segment_header(table_name REGCLASS, header_info CSTRING)

  Description: Updates the header of a segment-page table. This function is not supported in the current version.

- gs_redis_hashbucket_update_inverse_pointer(buckets TEXT, origin_dn_name TEXT, new_dn_name TEXT)

  Description: Updates the inverse pointers of buckets of this batch in all hash bucket tables in the current database. This function is not supported in the current version.

- gs_redis_local_get_inverse_pointer(buckets TEXT, origin_dn_name TEXT, new_dn_name TEXT)

  Description: Obtains the inverse pointer. This function is not supported in the current version.

- gs_redis_local_update_inverse_pointer(table_name TEXT, header_info TEXT, bucketlist TEXT)

  Description: Records Xlogs for updating inverse pointers. This function is not supported in the current version.

- gs_redis_local_set_hashbucket_frozenxid

  Description: Changes the **relfrozenxid64** value of the hash bucket table in the system catalog. This function is not supported in the current version.

- gs_redis_set_hashbucket_frozenxid(origin_group_id OID, target_group_id OID)

  Description: Changes the **relfrozenxid64** value of the hash bucket table in the system catalog on the new DN. This function is not supported in the current version.

- gs_redis_set_nextxid(xid BIGINT)

  Description: Modifies the **next_xid** value of a DN. This function is not supported in the current version.

- gs_redis_set_csn(csn BIGINT)

  Description: Changes the **next_csn** value of a DN. This function is not supported in the current version.

- gs_redis_check_bucket_flush(dn_array NAME[])

  Description: Determines whether all private buffers for RTO replay are flushed. This function is not supported in the current version.

- gs_redis_get_flush_page_lsn(isclean bool)

  Description: Queries the page refreshing information of bucket scale-out replay. This function is not supported in the current version.

- gs_redis_show_bucketxid(bucketid_list OIDVECTOR_EXTEND)

  Description: Queries the bucketxid corresponding to a specified bucket. This function is not supported in the current version.

- gs_redis_drop_bucket_files(origin_group_id OID, target_group_id OID)

  Description: Deletes the files that have been physically migrated from the source node. This function is not supported in the current version.

- gs_redis_local_drop_bucket_files(bucketlist CSTRING, bucketnum SMALLINT)

  Description: Deletes the corresponding bucket list. This function is not supported in the current version.

7.6.26.13 Other Functions

- plan_seed()

  Description: Obtains the seed value of the previous query statement (internal use).

  Return type: int

- pg_stat_get_env()

  Description: Obtains the environment variable information of the current node. Only users with the sysadmin or monitor admin permission can access the environment variable information.

  Return type: record

  Example:

```
gaussdb=# SELECT pg_stat_get_env();
                                                   pg_stat_get_env
-------------------------------------------------------------------------------------------------------------------------
----------------
 (sgnode,"localhost,XXX.XXX.XXX.XXX",28589,26000,/home/omm,/home/omm/data/
single_node,gs_log)
(1 row)
```

- pg_catalog.plancache_clean()

  Description: Clears the global plan cache that is not used on nodes.

  Return type: Boolean

- pg_catalog.plancache_status()

  Description: Displays information about the global plan cache on nodes. The information returned by the function is the same as that in **GLOBAL_PLANCACHE_STATUS**.

Return type: record

- textlen(text)

  Description: Provides the method of querying the logical length of text.

  Return type: int

- threadpool_status()

  Description: Displays the status of worker threads and sessions in the thread pool.

  Return type: record

- get_local_active_session()

  Description: Provides sampling records of the historical active sessions stored in the memory of the current node.

  Return type: record

- pg_stat_get_thread()

  Description: Provides status information about all threads on the current node. users with the sysadmin or monitor admin permission can view information about all threads, and common users can view only their own thread information.

  Return type: record

- pg_stat_get_sql_count()

  Description: Provides the counts of the SELECT, UPDATE, INSERT, DELETE, and MERGE INTO statements executed on the current node. Users with the sysadmin or monitor admin permission can view information about all users, and common users can view only their own statistics.

  Return type: record

- pg_stat_get_data_senders()

  Description: Provides detailed information about the data-copy sender thread active at the moment.

  Return type: record

- get_wait_event_info()

  Description: Provides detailed information about the wait event.

  Return type: record

- generate_wdr_report(begin_snap_id bigint, end_snap_id bigint, report_type cstring, report_scope cstring, node_name cstring)

  Description: Generates system diagnosis reports based on two snapshots. You need to run the command in the system database. By default, the initial user or users with the monadmin role permission can access the database. The result can be queried only in the system database but cannot be queried in the user database.

  Return type: record

  Table 7-95 generate_wdr_report parameter description

Parameter	Description	Range

begin_snap_id	Snapshot ID that starts the diagnosis report period.	N/A
end_snap_id	Snapshot ID that ends the diagnosis report period. By default, the value of **end_snap_id** is greater than that of **begin_snap_id**.	N/A
report_type	Specifies the type of the generated report.	• **summary** • **detail** • **all**: Both **summary** and **detail** are included.
report_scope	Specifies the scope for a report to be generated.	• **cluster**: database-level information • **node**: node-level information
node_name	When **report_scope** is set to **node**, set this parameter to the name of the corresponding node. (You can run the **select * from pg_node_env;** command to query the node name.) If **report_scope** is set to **cluster**, the value can be omitted, left blank, or **NULL**.	• **cluster**: omitted, empty, or **NULL**. • **node**: node name in GaussDB.

- create_wdr_snapshot()

  Description: Manually generates system diagnosis snapshots. This function requires the sysadmin permission.

  Return type: text

- kill_snapshot()

  Description: Kills the WDR snapshot backend thread. Users who call this function must have the SYSADMIN permission, the REPLICATION permission, or inherit the gs_role_replication permission of the built-in role.

  Return type: void

- generate_asp_report(start_time timestamp with time zone, end_time timestamp with time zone, slot_count bigint)

  Description: Generates ASP diagnosis reports based on timestamps, which can be accessed by the monadmin user. **slot_count** indicates the number of time segments for displaying metrics.

  Return type: text

- dbe_perf.get_active_session_profile(start_ts timestamp with time zone, end_ts timestamp with time zone, need_final boolean)

Description: Queries data in the ASP memory and disk based on timestamps. **need_final** specifies whether to query blocking information. The default value is **false**.

Return type: record

- capture_view_to_json(text,integer)

  Description: Saves the view result to the directory specified by GUC: **perf_directory**. If **is_crossdb** is set to **1**, the view is accessed once for all databases. If the value of **is_crossdb** is **0**, the current database is accessed only once. Only users with the sysadmin or monitor admin permission can execute this function.

  Return type: int

- reset_unique_sql(text,text,bigint)

  Description: Clears the unique SQL statements in the memory of the database node. (The sysadmin or monitor admin permission is required.)

  Return type: Boolean

Table 7-96 reset_unique_sql parameter description

Parameter	Type	Description
scope	text	Clearance scope type. The options are as follows: • **'GLOBAL'**: Clears all nodes. If the value is **'GLOBAL'**, this function can be executed only on the primary node. • **'LOCAL'**: Clears the current node.
clean_type	text	• **'BY_USERID'**: Unique SQL statements are cleared based on user IDs. • **'BY_CNID'**: Unique SQL statements are cleared based on primary node IDs. • **'ALL'**: All data is cleared.

Parameter	Type	Description
clean_value	int8	Clearance value corresponding to the clearance type. If the second parameter is set to **ALL**, the third parameter does not take effect and can be set to any value.

- wdr_xdb_query(db_name_str text, query text)

  Description: Provides the capability of executing local cross-database queries. For example, when connecting to the **testdb** database, only the initial user has the permission to access the table in the **test** database.

  ```
  SELECT col1 FROM wdr_xdb_query('dbname=test','SELECT col1 FROM t1') AS dd(col1 int);
  ```

  Return type: record

- pg_wlm_jump_queue(pid int)

  Description: Moves a task to the top of the queue of primary database node.

  Return type: Boolean

  - **true**: success
  - **false**: failure

- gs_wlm_switch_cgroup(pid int, cgroup text)

  Description: Moves a job to another Cgroup to change the job priority.

  Return type: Boolean

  - **true**: success
  - **false**: failure

- pv_session_memctx_detail(threadid tid, MemoryContextName text)

  Description: Records information about the memory context **MemoryContextName** of the thread **tid** into the *threadid*_**timestamp.log** file in the *$GAUSSLOG*/**gs_log/***${node_name}*/**dumpmem** directory. *threadid* can be obtained from *sessid* in the GS_SESSION_MEMORY_DETAIL view. In the officially released version, only **MemoryContextName** that is an empty string (two single quotation marks indicate that the input is an empty string) is accepted. In this case, all memory context information is recorded. Otherwise, no operation is performed. This function can be executed only by the administrator.

  Return type: Boolean

  - **true**: success
  - **false**: failure

- pg_shared_memctx_detail(MemoryContextName text)

  Description: Records information about the memory context **MemoryContextName** into the *threadid*_**timestamp.log** file in the *$GAUSSLOG*/**gs_log/***${node_name}*/**dumpmem** directory. This function is provided only for internal development and test personnel to debug in the DEBUG version. Calling this function in the officially released version does not involve any operation. Only the administrator can execute this function.

Return type: Boolean

- **true**: success
- **false**: failure

- local_aio_completer_stat()

  Description: Displays statistics about the AIO Completer thread in the instance.

  Return type: record

  Table 7-97 Return values of local_aio_completer_stat

Parameter	Type	Description
node_name	text	Name of the current instance.
tid	int8	ID of the AIO Completer thread.
thread_type	text	AIO Completer thread type (read or write).
aio_submitted_num	int8	Number of committed asynchronous I/O requests of the AIO Completer thread.
aio_completed_num	int8	Number of completed asynchronous I/O requests of the AIO Completer thread.
aio_incompleted_num	int8	Number of asynchronous I/O requests that are not completed of the AIO Completer thread.
slot_count_left	int8	Indicates the number of idle slots.

- local_aio_slot_usage_status()

  Description: Displays statistics about asynchronous I/O commit slots in the instance.

  Return type: record

  Table 7-98 Return values of local_aio_slot_usage_status

Parameter	Type	Description
nodename	text	Name of the current instance.
slot_id	int4	Slot ID.
slot_type	char	Slot type. The value can be **r** (read) or **w** (write).
status	bool	Slot usage status.
buffer_id	int8	Buffer ID corresponding to the slot.

Parameter	Type	Description
relfilenode_blocknum	text	Position of the physical page where the buffer corresponding to the slot is located.
lsn	int8	LSN corresponding to the page.
submitted_time	int8	Time when a page is committed asynchronously.
elapsed_time	int8	Waiting time of the page.

- gs_get_io_type()

  Description: Displays the I/O mode of the instance.

  Return type: text

  - **BIO**: The instance is running in BIO mode (ADIO is disabled).
  - **DIO**: The instance is running in DIO mode (ADIO is enabled).
  - **BIO->DIO (In progress)**: The current instance is switching from the BIO mode to the DIO mode.
- local_bgwriter_stat()

  Description: Displays the information about pages flushed by the bgwriter thread of this instance, number of pages in the candidate buffer chain, and buffer elimination information.

  Return type: record
- local_candidate_stat()

  Description: Displays the number of pages in the candidate buffer chain of this instance and buffer elimination information, including the normal buffer pool and segment buffer pool.

  Return type: record
- local_ckpt_stat()

  Description: Displays the information about checkpoints and flushing pages of the current instance.

  Return type: record
- local_double_write_stat()

  Description: Displays the doublewrite file status of the current instance.

  Return type: record

  Table 7-99 local_double_write_stat parameters

Parameter	Type	Description
node_name	text	Instance name.
curr_dwn	int8	Sequence number of the doublewrite file.
curr_start_page	int8	Start page for restoring the doublewrite file.

Parameter	Type	Description
file_trunc_num	int8	Number of times that the doublewrite file is reused.
file_reset_num	int8	Number of reset times after the doublewrite file is full.
total_writes	int8	Total number of I/Os of the doublewrite file.
low_threshold_writes	int8	Number of I/Os for writing doublewrite files with low efficiency (the number of I/O flushing pages at a time is less than 16).
high_threshold_writes	int8	Number of I/Os for writing doublewrite files with high efficiency (the number of I/O flushing pages at a time is more than 421).
total_pages	int8	Total number of pages that are flushed to the doublewrite file area.
low_threshold_pages	int8	Number of pages that are flushed with low efficiency.
high_threshold_pages	int8	Number of pages that are flushed with high efficiency.
file_id	int8	ID of the current doublewrite file.

- local_single_flush_dw_stat()

  Description: Displays the elimination of dual-write files on a single page in the instance.

  Return type: record

- local_pagewriter_stat()

  Description: Displays the page flushing information and checkpoint information of the current instance.

  Return type: record

- local_redo_stat()

  Description: Displays the replay status of the current standby instance.

  Return type: record

  Note: The returned replay status includes the current replay position and the replay position of the minimum restoration point.

- local_recovery_status()

  Description: Displays log flow control information about the primary and standby nodes.

  Return type: record

- gs_wlm_node_recover(boolean isForce)

  Description: Obtains top SQL query statement-level statistics recorded in the current memory. If the input parameter is not **0**, the information is cleared from the memory.

Return type: record

- gs_cgroup_map_ng_conf(group name)

  Description: Reads the Cgroup configuration file of a specified logical database.

  Return type: record

- gs_wlm_switch_cgroup(sess_id int8, cgroup name)

  Description: Switches the Cgroup of a specified session.

  Return type: record

- comm_client_info()

  Description: Queries information about active client connections of a single node.

  Return type: SETOF record

- pg_get_flush_lsn()

  Description: Returns the position of the Xlog flushed from the current node.

  Return type: text

- pg_get_sync_flush_lsn()

  Description: Returns the position of the Xlog flushed by the majority on the current node.

  Return type: text

- gs_create_log_tables()

  This function is unavailable in the current version.

- dbe_perf.get_global_full_sql_by_timestamp(start_timestamp timestamp with time zone, end_timestamp timestamp with time zone)

  Description: Obtains full SQL information at the database level. The result can be queried only in the system database but cannot be queried in the user database.

  Return type: record

  Table 7-100 dbe_perf.get_global_full_sql_by_timestamp parameter description

Parameter	Type	Description
start_timestamp	timestamp with time zone	Start point of the SQL start time range.
end_timestamp	timestamp with time zone	End point of the SQL start time range.

- dbe_perf.get_global_slow_sql_by_timestamp(start_timestamp timestamp with time zone, end_timestamp timestamp with time zone)

  Description: Obtains slow SQL information at the database level. The result can be queried only in the system database but cannot be queried in the user database.

  Return type: record

Table 7-101 dbe_perf.get_global_slow_sql_by_timestamp parameter description

Parameter	Type	Description
start_timestamp	timestamp with time zone	Start point of the SQL start time range.
end_timestamp	timestamp with time zone	End point of the SQL start time range.

- statement_detail_decode(detail text, format text, pretty boolean)

  Description: Parses the **details** column in a full or slow SQL statement. The result can be queried only in the system database but cannot be queried in the user database.

  Return type: text

  Table 7-102 statement_detail_decode parameter description

Parameter	Type	Description
detail	text	Set of events generated by the SQL statement (unreadable).
format	text	Parsing output format. The value is **plaintext**.
pretty	Boolean	Specifies whether to display the text in pretty format when **format** is set to **plaintext**. The options are as follows: • The value **true** indicates that events are separated by **\n**. • The value **false** indicates that events are separated by commas (,).

- pgxc_get_csn(tid)

  Description: Returns the transaction commit sequence number (CSN) corresponding to a given transaction ID.

  Return type: int8

- pgxc_get_csn(tid, bucketid)

  Description: The hash bucket table is not supported in the current version in centralized mode. An error is reported when the function is called.

- pg_control_system()

  Description: Returns the status of the system control file.

  Return type: SETOF record

- pg_control_checkpoint()

  Description: Returns the system checkpoint status.

  Return type: SETOF record

- get_prepared_pending_xid()

  Description: Returns nextxid when restoration is complete.

  Parameter: nan

  Return type: text
- pg_clean_region_info()

  Description: Clears the regionmap.

  Parameter: nan

  Return type: character varying
- pg_get_replication_slot_name

  Description: Obtains the slot name.

  Parameter: nan

  Return type: text
- pg_get_running_xacts()

  Description: Obtains running xact.

  Parameter: nan

  Return type: handle integer, gxid xid, state tinyint, node text, xmin xid, vacuum boolean, timeline bigint, prepare_xid xid, pid bigint, and next_xid xid
- pg_get_variable_info()

  Description: Obtains the shared memory variable *cache*.

  Parameter: nan

  Return type: node_name text, nextOid oid, nextXid xid, oldestXid xid, xidVacLimit xid, oldestXidDB oid, lastExtendCSNLogpage xid, startExtendCSNLogpage xid, nextCommitSeqNo xid, latestCompletedXid xid, and startupMaxXid xid
- pg_get_xidlimit()

  Description: Obtains transaction ID information from the shared memory.

  Parameter: nan

  Return type: nextXid xid, oldestXid xid, xidVacLimit xid, xidWarnLimit xid, xidStopLimit xid, xidWrapLimit xid, and oldestXidDB oid
- pg_relation_compression_ratio()

  Description: Queries the compression rate of a table. By default, **1.0** is returned.

  Parameter: text

  Return type: real
- pg_relation_with_compression()

  Description: Queries whether a table is compressed.

  Parameter: text

  Return type: Boolean
- pg_stat_file_recursive()

  Description: Lists all files in the path.

  Parameter: location text

  Return type: path text, filename text, size bigint, and isdir boolean

- pg_stat_get_activity_for_temptable()

  Description: Returns records of backend threads related to the temporary table.

  Parameter: nan

  Return type: datid oid, timelineid integer, tempid integer, and sessionid bigint
- pg_stat_get_activity_ng()

  Description: Returns records of backend threads related to nodegroup.

  Parameter: pid bigint

  Return type: datid oid, pid bigint, sessionid bigint, and node_group text
- pg_stat_get_cgroup_info()

  Description: Returns Cgroup information.

  Parameter: nan

  Return type: cgroup_name text, percent integer, usage_percent integer, shares bigint, usage bigint, cpuset text, relpath text, valid text, and node_group text
- pg_stat_get_realtime_info_internal()

  Description: Returns real-time information. Currently, this API is unavailable. **FailedToGetSessionInfo** is returned.

  Parameter: oid, oid, bigint, cstring, and oid

  Return type: text
- pg_test_err_contain_err()

  Description: Tests the error type and return information.

  Parameter: integer

  Return type: void
- get_global_user_transaction()

  Description: Returns transaction information about each user on all nodes.

  Return type: node_name name, usename name, commit_counter bigint, rollback_counter bigint, resp_min bigint, resp_max bigint, resp_avg bigint, resp_total bigint, bg_commit_counter bigint, bg_rollback_counter bigint, bg_resp_min bigint, bg_resp_max bigint, bg_resp_avg bigint, and bg_resp_total bigint
- pg_collation_for()

  Description: Returns the sorting rule corresponding to the input parameter string.

  Parameter: any (Explicit type conversion is required for constants.)

  Return type: text
- pgxc_unlock_for_sp_database(name Name)

  This function is unavailable in the current version.
- pgxc_lock_for_sp_database(name Name)

  This function is unavailable in the current version.
- copy_error_log_create()

  Description: Creates the error table (**public.pgxc_copy_error_log**) required for creating the **COPY FROM** error tolerance mechanism.

  Return type: Boolean

NOTE

- This function attempts to create the **public.pgxc_copy_error_log** table. For details about the table, see **Table 7-103**.
- In addition, it creates a B-tree index on the **relname** column and executes **REVOKE ALL on public.pgxc_copy_error_log FROM public** to manage permissions on the error table (the permissions are the same as those of the COPY statement).
- **public.pgxc_copy_error_log** is a row-store table. Therefore, this function can be executed and COPY error tolerance is available only when row-store tables can be created in the database instance. Note that after the GUC parameter **enable_hadoop_env** is enabled, row-store tables cannot be created in the database instance (the default value is **off** for GaussDB).
- Same as the error table and the COPY statement, the function requires sysadmin or higher permissions.
- If the **public.pgxc_copy_error_log** table or the **copy_error_log_relname_idx** index exists before the function creates it, the function will report an error and roll back.

Table 7-103 Error table public.pgxc_copy_error_log

Column	Type	Description
relname	character varying	Table name in the form of *Schema name*.*Table name*
begintime	timestamp with time zone	Time when a data format error was reported
filename	character varying	Name of the source data file where a data format error occurs
lineno	bigint	Number of the row where a data format error occurs in a source data file
rawrecord	text	Raw record of a data format error in the source data file
detail	text	Error details

- dynamic_func_control(scope text, function_name text, action text, "{params}" text[])

  Description: Dynamically enables built-in functions. Currently, only full SQL statements can be dynamically enabled.

  Return type: record

Table 7-104 Parameter description of dynamic_func_control

Parameter	Type	Description
scope	text	Scope where the function is to be dynamically enabled. Currently, only **LOCAL** is supported.
function_name	text	Function name. Currently, only **STMT** is supported.

Parameter	Type	Description
action	text	When **function_name** is set to **STMT**, the value of **action** can only be **TRACK**, **UNTRACK**, **LIST**, or **CLEAN**. • **TRACK**: records the full SQL information of normalized SQL statements. • **UNTRACK**: cancels the recording of full SQL information of normalized SQL statements. • **LIST**: lists normalized SQL information that is recorded in the current track. • **CLEAN**: cleans normalized SQL information that is recorded in the current track.
params	text[]	When **function_name** is set to **STMT**, the parameters corresponding to different actions are set as follows: • **TRACK**: **'{"Normalized SQLID", "L0/L1/L2"}'** • **UNTRACK**: **'{"Normalized SQLID"}'** • **LIST**: **'{}'** • **CLEAN**: **'{}'**

- gs_parse_page_bypath(path text, blocknum bigint, relation_type text, read_memory boolean)

  Description: Parses a specified table page and returns the path for storing the parsed content.

  Return type: text

  Note: Only the system administrator or O&M administrator can execute this function.

Table 7-105 gs_parse_page_bypath parameters

Parameter	Type	Description
path	text	• For an ordinary table or segment-page table, the relative path is *Tablespace name*/*Database OID*/*Relfilenode of the table (physical file name)*. For example, **base/16603/16394**. • For the visibility map of an ordinary table, the relative path is *Tablespace name*/*Database OID*/*Visibility map of the ordinary table*. For example, **base/16603/16394_vm**. • For clog files, the parsed content is stored in the **pg_clog** directory. For example: 000000000000. • For csnlog files, the parsed content is stored in the **pg_csnlog** directory. For example: 000000000000. • For undo record files, the relative path is **undo/UNDOPERSISTENCE/zondid.segno** in the **undo** directory. Example: undo/permanent/00000.0000009. • For undo meta files, the relative path is **undo/UNDOPERSISTENCE/zondid.meta.segno** in the **undo** directory. Example: undo/permanent/00000.meta.0000004. • You can run the **pg_relation_filepath(table_name text)** command to query the relative path of the table file. To obtain the path of the partitioned table, view the **pg_partition** system catalog and call **pg_partition_filepath(partition_oid)**. • Valid path formats are as follows: – global/relNode. – base/dbNode/relNode. – pg_tblspc/spcNode/version_dir/dbNode/relNode.

Parameter	Type	Description
blocknum	bigint	• **–1**: Information about all blocks (forcibly parsed from disks) • **0**–*MaxBlockNumber*: Information about the corresponding block • For B-tree/UB-tree indexes, **0** indicates the index meta-page. • For undo record files, the logical block number is used. The block number of the corresponding file is greater than or equal to **segno** x 128 and less than (**segno** + 1) x 128. • For undo meta files, the logical block number is used. The block number of the corresponding file is greater than or equal to **segno** x 4 and less than (**segno** + 1) x 4.
relation_type	text	• **heap**: Astore table • **uheap**: Ustore table • **btree**: B-tree index • **ubtree**: UB-tree index • **vm**: visibility map of the Astore ordinary table • **clog** (commit log): transaction status log • **csnlog** (commit sequence number log): snapshot timestamp log • **undo_slot**: transaction slot information • **undo_record**: undo record information • **indexurq**: indexurq page • **gsivfflat_index**: gsivfflat vector index • **gsdiskann_index**: gsdiskann vector index
read_memory	Boolean	• **false**: The system parses the page from the disk file. • **true**: The system attempts to parse the page from the shared buffer. If the page does not exist in the shared buffer, the system parses the page from the disk file.

Example:

```
# Parse the information of all pages in the B-tree index file.
# Before calling the function, ensure that the file path exists based on the parameter description.
gaussdb=# SELECT gs_parse_page_bypath('base/16603/16394', -1, 'btree',  false);
                        gs_parse_page_bypath
---------------------------------------------------------------------
 /gs_log_dir/dump/1663_16603_16394_-1.page
(1 row)
```



```
# Parse the visibility result of all blocks in the visibility map file.
gaussdb=# SELECT gs_parse_page_bypath('base/12828/16771_vm', -1, 'vm', false);
                      gs_parse_page_bypath
-------------------------------------------------------------------------
 /gs_log_dir/dump/1663_12828_16771_-1_vm.page
(1 row)

# Parse the commit log of block 0 in the Clog file.
gaussdb=# SELECT gs_parse_page_bypath('000000000000', 0, 'clog', false);
                     gs_parse_page_bypath
--------------------------------------------------------------------
 /gs_log_dir/dump/000000000000.clog
(1 row)
```

The following is an example of an exception error:

```
# An error is reported when the value of the block number exceeds the value range.
gaussdb=# SELECT gs_parse_page_bypath('base/12828/16777', -10, 'heap', false);
ERROR:  Blocknum should be between -1 and 4294967294.
CONTEXT:  referenced column: gs_parse_page_bypath
```

- gs_xlogdump_lsn(start_lsn text, end_lsn text)

  Description: Parses Xlogs within the specified LSN range and returns the path for storing the parsed content. You can use **pg_current_xlog_location()** to obtain the current Xlog position.

  Return type: text

  Parameters: LSN start position and LSN end position

  Note: Only the system administrator or O&M administrator can execute this function.

- gs_xlogdump_xid(c_xid xid)

  Description: Parses Xlogs of a specified XID and returns the path for storing the parsed content. You can use txid_current() to obtain the current XID.

  Parameter: XID

  Return type: text

  Note: Only the system administrator or O&M administrator can execute this function.

- gs_xlogdump_tablepath(path text, blocknum bigint, relation_type text)

  Description: Parses logs corresponding to a specified table page and returns the path for storing the parsed content.

  Return type: text

  Note: Only the system administrator or O&M administrator can execute this function.

Table 7-106 gs_xlogdump_tablepath parameters

Parameter	Type	Description
path	text	• For an ordinary table or segment-page table, the relative path is *Tablespace name*/*Database OID*/*Relfilenode of the table (physical file name)*. For example, **base/16603/16394**. • You can run the **pg_relation_filepath(table_name text)** command to query the relative path of the table file. To obtain the path of the partitioned table, view the pg_partition system catalog and call pg_partition_filepath(partition_oid). • Valid path formats are as follows: – global/relNode – base/dbNode/relNode – pg_tblspc/spcNode/version_dir/dbNode/relNode
blocknum	bigint	• **–1**: Information about all blocks (forcibly parsed from disks) • **0**–*MaxBlockNumber*: Information about the corresponding block
relation_type	text	• **heap**: Astore table • **uheap**: Ustore table • **btree**: B-tree index • **ubtree**: UB-tree index • **segment**: segment-page table

- gs_xlogdump_parsepage_tablepath(path text, blocknum bigint, relation_type text,read_memory boolean)

  Description: Parses the specified table page and logs corresponding to the table page and returns the path for storing the parsed content. It can be regarded as one execution of **gs_parse_page_bypath** and **gs_xlogdump_tablepath**. The prerequisite for executing this function is that the table file exists. To view logs of deleted tables, call **gs_xlogdump_tablepath**.

  Return type: text

  Note: Only the system administrator or O&M administrator can execute this function.

Table 7-107 gs_xlogdump_parsepage_tablepath parameters

Parameter	Type	Description
path	text	• For ordinary tables, the relative path is *Tablespace name*/*Database OID*/*Relfilenode of the table (physical file name)*. For example, **base/16603/16394**. • You can run the **pg_relation_filepath(table_name text)** command to query the relative path of the table file. To obtain the path of the partitioned table, view the pg_partition system catalog and call pg_partition_filepath(partition_oid). • Valid path formats are as follows: – global/relNode. – base/dbNode/relNode. – pg_tblspc/spcNode/version_dir/dbNode/relNode.
blocknum	bigint	• **–1**: information about all blocks (forcibly parsed from disks) • **0**–*MaxBlockNumber*: information about the corresponding block
relation_type	text	• **heap**: Astore table • **uheap**: Ustore table • **btree**: B-tree index • **ubtree**: UB-tree index • **indexurq**: indexurq page
read_memory	Boolean	• **false**: The system parses the page from the disk file. • **true**: The system attempts to parse the page from the shared buffer. If the page does not exist in the shared buffer, the system parses the page from the disk file.

- gs_index_verify(oid oid, uint32 blkno)

  Description: Checks whether the sequence of keys on the UB-tree index page or index tree is correct.

  Return type: record

Table 7-108 gs_index_verify parameters

Parameter	Type	Description
oid	oid	• Index file relfilenode, which can be queried using **select relfilenode from pg_class where relname='***Index file name***'**.
blkno	uint32	• **0**: indicates that all pages in the index tree are checked. • If the value is greater than 0, the index page whose page code is equal to the value of **blkno** is checked.

- gs_index_recycle_queue(Oid oid, int type, uint32 blkno)

  Description: Parses the UB-tree index recycling queue information.

  Return type: record

  Table 7-109 gs_index_recycle_queue parameters

Parameter	Type	Description
oid	oid	• Index file relfilenode, which can be queried using **select relfilenode from pg_class where relname='***Index file name***'**.
type	int	• **0**: The entire queue to be recycled is parsed. • **1**: The entire empty page queue is parsed. • **2**: A single page is parsed.
blkno	uint32	• ID of the recycling queue page. This parameter is valid only when **type** is set to **2**. The value of **blkno** ranges from 1 to 4294967294.

- gs_stat_wal_entrytable(int64 idx)

  Description: Exports the content of the write-ahead log insertion status table in the Xlog.

  Return type: record

  Table 7-110 gs_stat_wal_entrytable parameters

Category	Parameter	Type	Description
Input parameter	idx	int64	• **–1**: queries all elements in an array. • **0**–*Maximum value*: content of a specific array element.

Category	Parameter	Type	Description
Output parameter	idx	uint64	Records the indexes in the corresponding array.
Output parameter	endlsn	uint64	Records the LSN label.
Output parameter	lrc	int32	Records the corresponding LRC.
Output parameter	status	uint32	Determines whether the Xlog corresponding to the current entry has been completely copied to the WAL buffer. • **0**: Not copied. • **1**: Copied

- gs_walwriter_flush_position()
  Description: Outputs the refresh position of write-ahead logs.
  Return type: record

  Table 7-111 gs_walwriter_flush_position parameters

Category	Parameter	Type	Description
Output parameter	last_flush_status_entry	int32	Index obtained after the Xlog flushes the tblEntry of the last flushed disk.
Output parameter	last_scanned_lrc	int32	LRC obtained after the Xlog flushes the last tblEntry scanned last time.
Output parameter	curr_lrc	int32	Latest LRC usage in the WALInsertStatusEntry status table. The LRC indicates the LRC value corresponding to the WALInsertStatusEntry when the next Xlog record is written.

Category	Parameter	Type	Description
Output parameter	curr_byte_pos	uint64	The latest Xlog position after the Xlog is written to the WAL file, which is also the next Xlog insertion point.
Output parameter	prev_byte_size	uint32	Length of the previous Xlog record.
Output parameter	flush_result	uint64	Position of the current global Xlog flush.
Output parameter	send_result	uint64	Xlog sending position on the current host.
Output parameter	shm_rqst_write_pos	uint64	The write position of the LogwrtRqst request in the XLogCtl recorded in the shared memory.
Output parameter	shm_rqst_flush_pos	uint64	The flush position of the LogwrtRqst request in the XLogCtl recorded in the shared memory.
Output parameter	shm_result_write_pos	uint64	The write position of the LogwrtResult request in the XLogCtl recorded in the shared memory.
Output parameter	shm_result_flush_pos	uint64	The flush position of the LogwrtResult request in the XLogCtl recorded in the shared memory.
Output parameter	curr_time	text	Current time.

- gs_walwriter_flush_stat(int operation)

  Description: Collects statistics on the frequency of writing and synchronizing write-ahead logs, data volume, and Xlog file information.

  Return type: record

Table 7-112 gs_walwriter_flush_stat parameters

Category	Parameter	Type	Description
Input parameter	operation	int	• **–1** (default): disables the statistics function. • **0**: enables the statistics function. • **1**: queries statistics. • **2**: resets statistics.
Output parameter	write_times	uint64	Number of times that the Xlog calls the write API.
Output parameter	sync_times	uint64	Number of times that the Xlog calls the sync API.
Output parameter	total_xlog_sync_bytes	uint64	Total number of backend thread requests for writing data to Xlogs.
Output parameter	total_actual_xlog_sync_bytes	uint64	Total number of Xlogs that call the sync API for disk flushing.
Output parameter	avg_write_bytes	uint32	Number of Xlogs written each time the XLogWrite API is called.
Output parameter	avg_actual_write_bytes	uint32	Number of Xlogs written each time the write API is called.
Output parameter	avg_sync_bytes	uint32	Average number of Xlogs for each synchronization request.
Output parameter	avg_actual_sync_bytes	uint32	Actual Xlog amount of disk flushing by calling sync each time.
Output parameter	total_write_time	uint64	Total time of calling the write operation (unit: μs).

Category	Parameter	Type	Description
Output parameter	total_sync_time	uint64	Total time for calling the sync API (unit: μs).
Output parameter	avg_write_time	uint32	Average time for calling the write API each time (unit: μs).
Output parameter	avg_sync_time	uint32	Average time for calling the sync API each time (unit: μs).
Output parameter	curr_init_xlog_segno	uint64	ID of the latest Xlog segment file.
Output parameter	curr_open_xlog_segno	uint64	ID of the Xlog segment file that is being written.
Output parameter	last_reset_time	text	Time when statistics were last collected.
Output parameter	curr_time	text	Current time.

- gs_catalog_attribute_records()

  Description: Returns the definition of each field in a specified system catalog. Only ordinary system catalogs whose OIDs are less than 10000 are supported. Indexes and TOAST tables are not supported.

  Parameter: OID of the system catalog

  Return type: record

- gs_comm_proxy_thread_status()

  Description: Collects statistics on data packets sent and received by the proxy communications library **comm_proxy** when a user-mode network is configured for the database instance.

  Parameter: nan

  Return type: record

> NOTE
>
> The query result of this function is displayed only when the user-mode network is deployed in a centralized environment and **enable_dfx in comm_proxy_attr** is set to **true**. In other scenarios, an error message is displayed, indicating that queries are not supported.

- pg_ls_tmpdir()

  Description: Returns the name, size, and last modification time of each file in the temporary directory (**pgsql_tmp**) of the default tablespace.

  Parameter: nan

  Return type: record

  Note: Only the system administrator or monitor administrator can execute this function.

Category	Parameter	Type	Description
Output parameter	name	text	File name.
Output parameter	size	int8	File size (unit: byte).
Output parameter	modification	timestamptz	Last file modification time.

- pg_ls_tmpdir(oid)

  Description: Returns the name, size, and last modification time of each file in the temporary directory (**pgsql_tmp**) of the specified tablespace.

  Parameter: oid

  Return type: record

  Note: Only the system administrator or monitor administrator can execute this function.

Category	Parameter	Type	Description
Input parameter	oid	oid	Tablespace ID.
Output parameter	name	text	File name.
Output parameter	size	int8	File size (unit: byte).
Output parameter	modification	timestamptz	Last file modification time.

- pg_ls_waldir()

  Description: Returns the name, size, and last modification time of each file in the WAL directory.

Parameter: nan

Return type: record

Note: Only the system administrator or monitor administrator can execute this function.

Category	Parameter	Type	Description
Output parameter	name	text	File name.
Output parameter	size	int8	File size (unit: byte).
Output parameter	modification	timestamptz	Last file modification time.

- gs_stat_anti_cache()

  Description: Returns AntiCache statistics.

  Parameter: nan

  Return type: record

Category	Parameter	Type	Description
Output parameter	table_nums	int8	Number of partitions.
Output parameter	table_init_size	int8	Initial size of a partition.
Output parameter	table_max_size	int8	Upper limit of the partition size.
Output parameter	anti_cache_upper_limit_size	int8	Upper limit of the AntiCache size.
Output parameter	anti_cache_size	int8	Real-time AntiCache size.
Output parameter	anti_cache_max_table_size	int8	Real-time maximum partition size.
Output parameter	anti_cache_min_table_size	int8	Real-time minimum partition size.

- gs_stat_vlog_buffer()

  Description: Returns verifyLog buffer statistics.

  Parameter: nan

  Return type: record

Category	Parameter	Type	Description
Output parameter	vbuffer_write_offset	int8	Vlog write location.
Output parameter	vbuffer_flushed_offset	int8	Vlog disk flushing location.
Output parameter	max_vbuffer_flushed_value	int8	Maximum number of bytes for vlog flushing.
Output parameter	min_vbuffer_flushed_vlaue	int8	Minimum number of bytes for vlog flushing.
Output parameter	ave_vbuffer_flushed_vlaue	int8	Average number of bytes for vlog flushing.
Output parameter	vbuffer_flush_latency	int8	Vlog refresh delay.

- gs_stat_vlog_related_io()

  Description: Returns I/O statistics about read and write operations on the verifylog file.

  Parameter: nan

  Return type: record

Category	Parameter	Type	Description
Output parameter	read_data_iops	float8	Number of pages read from files per second.
Output parameter	vlog_read_iops	float8	Number of pages read from vlogs per second.
Output parameter	vlog_write_iops	float8	Number of pages written to vlogs per second.

- gs_write_term_log(void)

  Description: Writes a log to record the current **term** value of a DN. The standby DN returns **false**. After the data is successfully written to the primary DN, **true** is returned.

  Return type: Boolean

- gs_stat_space(bool init)

  Description: Queries the status of extended pages when the INSERT operation is performed on the Ustore table.

  Return type: record

Category	Parameter	Type	Description
Input parameter	init	bool	Specifies whether to reset the statistics.
Output parameter	access_func	int8	Total number of access times of the relation_get_buffer_for_utuple API.
Output parameter	cache_blk	int8	Number of times that the relation_get_buffer_for_utuple API obtains buffers.
Output parameter	cache_succ	int8	Number of times that the relation_get_buffer_for_utuple API successfully obtains buffers.
Output parameter	nblk_first	int8	Number of times that relation_get_buffer_for_utuple obtains **nblocks-1** for the first time.
Output parameter	nblk_first_succ	int8	Number of times that relation_get_buffer_for_utuple obtains **nblocks-1** successfully for the first time.
Output parameter	nblk_second	int8	Number of times that relation_get_buffer_for_utuple obtains **nblocks-1** for the second time.
Output parameter	nblk_second_succ	int8	Number of times that relation_get_buffer_for_utuple obtains **nblocks-1** successfully for the second time.
Output parameter	fsm_first	int8	Number of times that FSM is accessed for the first time.
Output parameter	fsm_first_success	int8	Number of times that FSM is accessed successfully for the first time.
Output parameter	fsm_rewrite	int8	Number of FSM writeback times.
Output parameter	fsm_second	int8	Number of times that FSM is accessed for the second time.
Output parameter	fsm_second_sucess	int8	Number of times that FSM is accessed successfully for the second time.

Category	Parameter	Type	Description
Output parameter	prune_count	int8	Number of prune times in relation_get_buffer_for_utuple.
Output parameter	prune_space	int8	Total prune space in relation_get_buffer_for_utuple.
Output parameter	coprune_count	int8	Number of times that joint prune is performed.
Output parameter	coprune_scan_blocks	int8	Total number of pages scanned for joint prune.
Output parameter	coprune_prune_count	int8	Number of joint prune times.
Output parameter	coprune_prune_space	int8	Total space for joint prune.
Output parameter	con_extend_count	int8	Number of concurrent extended pages.
Output parameter	con_extend_time	int8	Total time for concurrent page extension.
Output parameter	single_extend_time	int8	Number of single-page extension times.

Example:

```
gaussdb=# SELECT * FROM gs_stat_space(false);
 access_func | cache_blk | cache_succ | nblk_first | nblk_first_succ | nblk_sencond | nblk_sencond_succ |
fsm_first | fsm_first_success | fsm_rewrite | fsm_
second | fsm_second_success | prune_count | prune_space | coprune_count | coprune_scan_blocks |
coprune_prune_count | coprune_prune_space | con_extend_count
 | con_extend_time | single_extend_count
-------------+-----------+------------+------------+-----------------+--------------+-------------------+-----------
+-------------------+-------------+-----
-------+-------------------+-------------+-------------+---------------+--------------------+--------------------
+--------------------+-----------------
-+-----------------+--------------------
    10082397 |  10082397 |   10082235 |        88 |             78 |       47021 |            47021 |       162
|              74 |      360996 |
360996 |            270948 |       6711 |          0 |        45497 |            222619 |                0
|                 0 |            3675
 |       25542884 |             26791
(1 row)
```

Note: Pay special attention to the value of **cache_succ**. If it is small, the system cache is invalid. If the value of **prune_space** is small, the Ustore data page cleaning mechanism may be faulty. If the value of **con_extend_time** is too large, the Ustore concurrent page extension takes a long time.

- gs_index_dump_read(int8 reset, text out_type)

  Description: Queries the buffer read information generated in the cyclic queue when an index is used to obtain a new page. The buffer read information traverses leaf pages from left to right using the same key as the index page.

  Return type: record

Category	Parameter	Type	Description
Input parameter	reset	int8	• **0**: The statistics are reset to the initial value 0 and the statistics are collected again. • **1**: The current statistics are displayed.
Input parameter	out_type	text	• **urq**: outputs statistics about cyclic queues. • **ubtree**: outputs statistics on index pages. • **all**: outputs all statistics about cyclic queues and index pages.
Output parameter	relfilenode	oid	Index relfilenode corresponding to the maximum buffer read value.
Output parameter	max_count	int8	Maximum buffer read value.
Output parameter	ave_count	int8	Average buffer read value.

NOTE

- Currently, this API supports only Ustore index tables.
- When this API is executed, the reset operation is performed to clear all records and set all records to **0**. If you query the information again, the query result is always 0 until the information is collected next time. The following are query examples:

```
gaussdb=# SELECT * FROM gs_index_dump_read(0, 'all');
relfilenode | max_count | ave_count
--------------+-------------+-------------
|                   |
(1 row)
gaussdb=# SELECT * FROM gs_index_dump_read(1, 'all');
relfilenode | max_count | ave_count
--------------+-------------+-------------
0 |                 0 |                 0
0 |                 0 |                 0
(2 rows)
```

- gs_redo_upage(directory_path text, backup_path text, blocknum bigint, relation_type text, xlog_path text, lsn text)

  Description: Redoes a specific Ustore data page that is backed up to a specified LSN and verifies the page in this process. If a damaged page is detected, the page is flushed to the disk and the disk flushing path, page LSN, and damage information is returned; otherwise, the page is redone to the specified LSN and flushed to the disk, and the result is returned. Only system administrators and O&M administrators can execute this function.

  Return type: record

Category	Parameter	Type	Description
Input parameter	directory_path	text	Specifies the directory for storing the backup file.
Input parameter	backup_path	text	Relative path of the backup table file, which is combined with the directory where the backup file is located to form the complete path of the table file, for example, **base/15635/12488**. If the backup file does not exist, set this parameter to null.
Input parameter	blocknum	bigint	**0** to *MaxBlockNumber*: block number of the corresponding page.
Input parameter	relation_type	text	• **uheap**: Ustore data page • **ubtree**: Ustore index page • **indexurq**: URQ page • **undo_record**: undo record page • **undo_slot**: transaction slot page
Input parameter	xlog_path	text	Absolute path of the archive log directory.

Category	Parameter	Type	Description
Input parameter	lsn	text	The LSN consists of two hexadecimal numbers (32 bits each), which are separated by a slash (/), for example, 2/962D1DF8. If the value is **0**, the latest version is used.
Output parameter	output_filename	text	Path and name of the file to be flushed to the disk.
Output parameter	output_lsn	text	LSN of the last page redo.
Output parameter	corruption_desc	text	Page damage description.

- gs_xlogdump_bylastlsn(last_lsn text, blocknum bigint, relation_type text)

  Description: Inputs a page LSN and block number, parses the WAL corresponding to the LSN, obtains the last LSN of the corresponding block number, continues parsing until the last LSN is 0 or the WAL of an earlier version has been reused and recycled, and flushes the parsed log to a specified path. Only system administrators or O&M administrators can execute this function. This system function cannot be called by the standby node.

  Return type: text

Category	Parameter	Type	Description
Input parameter	last_lsn	text	Parses the LSN of a specified page in hexadecimal notation, for example, 12BA/32CDEDDD. The LSN can be obtained using a page parsing tool (such as **gs_parse_page_bypath**).
Input parameter	blocknum	bigint	Specifies the logical block number of a page. Value range: **-1** to *MaxBlockNumber*. If the block number is set to **-1**, the default block number is obtained from the WAL.
Input parameter	relation_type	text	Specifies the type of the page to be parsed. Valid value: **uheap**, **ubtree**, **heap**, **btree**, **undo_record**, and **undo_slot**.
Output parameter	output_filepath	text	Specifies the path for flushing WAL parsing results to disks.

Example:

```
# Obtain the page LSN.
# Before calling the function, ensure that the file path exists based on the parameter description.
gaussdb=# SELECT * FROM gs_parse_page_bypath('base/15833/16768', 0, 'uheap', false);
                      output_filepath
-------------------------------------------------------------------
 /data1/database/cluster/primary/data/1663_15833_16768_0.page
(1 row)
gaussdb=# SELECT * FROM gs_xlogdump_bylastlsn('0/4593570', -1, 'uheap');
                      output_filepath
-------------------------------------------------------------------
 /data1/database/cluster/primary/data/gs_log/dump/4593570_-1.xlog
(1 row)
gaussdb=# SELECT * FROM gs_xlogdump_bylastlsn('0/4593570', 0, 'ubtree');
ERROR:  The input lsn 0/4593570 related xlog is not ubtree.
```

- gs_shared_storage_flush_stat(int operation)

  Description: Collects statistics on the amount of written data, write speed, write time, amount of read data, read speed, and read time of shared disks.

  Return type: record

Category	Parameter	Type	Description
Input parameter	operation	int	• **-1**: disables the statistics function. • **0** (default): enables the statistics function. • **1**: queries statistics. • **2**: resets statistics. After the statistics function is disabled and then enabled again, the previous statistics are cleared.
Output parameter	stat_switch	bool	Specifies whether the statistics function is enabled.
Output parameter	write_times	uint64	Number of times that the shared_storage_xlog_copy thread calls the dorado_write_xlog API.
Output parameter	avg_write_bytes	uint32	Amount of Xlogs to be written each time the dorado_write_xlog API is called, in byte.
Output parameter	avg_actual_write_bytes	uint32	Actual amount of Xlogs to be written each time the dorado_write_xlog API is called, in byte.
Output parameter	total_write_time	uint64	Total time for calling the dorado_write_xlog API, in μs.

Category	Parameter	Type	Description
Output parameter	avg_write_time	uint32	Average time for calling the dorado_write_xlog API each time, in µs.
Output parameter	avg_write_speed	uint32	Average speed of invoking the dorado_write_xlog API to write Xlogs, in KB/s.
Output parameter	avg_actual_write_speed	uint32	Actual average speed of invoking the dorado_write_xlog API to write Xlogs, in KB/s.
Output parameter	total_write_sleep_time	uint64	Total sleep time of the shared_storage_xlog_copy thread.
Output parameter	read_times	uint64	Number of times that the walreceiver thread calls the dorado_read_xlog API.
Output parameter	avg_read_bytes	uint32	Amount of Xlogs to be written each time the dorado_read_xlog API is called, in byte.
Output parameter	avg_actual_read_bytes	uint32	Actual amount of Xlogs to be written each time the dorado_read_xlog API is called, in byte.
Output parameter	total_read_time	uint64	Total time for calling the dorado_read_xlog API, in µs.
Output parameter	avg_read_time	uint32	Average time for calling the dorado_read_xlog API each time, in µs.
Output parameter	avg_read_speed	uint32	Average speed of invoking the dorado_read_xlog API to write Xlogs, in KB/s.
Output parameter	avg_actual_read_speed	uint32	Actual average speed of invoking the dorado_read_xlog API to write Xlogs, in KB/s.
Output parameter	total_read_sleep_time	uint64	Total sleep time of the walreceiver thread.
Output parameter	stat_start_time	timestamptz	Start time of the statistics. If the statistics are not reset, the value is the start time of GaussDB. If the statistics are reset, the value is the last reset time.

Category	Parameter	Type	Description
Output parameter	stat_end_time	timestampt z	End time of the current statistics collection. If the statistics collection function is enabled, the value is the current time. If the statistics collection function is disabled, the value is the last time when the statistics collection function is disabled.

Example:

```
gaussdb=# SELECT * FROM gs_shared_storage_flush_stat(1);
 stat_switch | write_times | avg_write_bytes | avg_actual_write_bytes | total_write_time |
avg_write_time | avg_write_speed | avg_actual_write_speed | total_write_sleep_time | read_times | avg
_read_bytes | avg_actual_read_bytes | total_read_time | avg_read_time | avg_read_speed |
avg_actual_read_speed | total_read_sleep_time |         stat_start_time         |          stat_end_time

-------------+-------------+-----------------+------------------------+------------------+----------------
+----------------+----------------------+-----------------------+------------+----
------------+----------------------+----------------+---------------+----------------+-----------------------
+-----------------------+-------------------------------+-----------------------
-------
 t           |       43644 | 8624            | 8624                   |        263922553 | 6047           | 1392           |
1392                   |            26918503404 |       7451 | 140
13          | 14013                |          5321082 | 714           | 19163          | 19163
|                    0 | 2023-12-18 07:18:39.766855+08 | 2023-12-18 16:29:22.757
535+08
(1 row)
```

- dbe_perf.get_full_sql_by_parent_id_and_timestamp(parent_id bigint, start_timestamp timestamp with time zone, end_timestamp timestamp with time zone)

  Description: Obtains the records of database-level full SQL statements and their substatements that execute a stored procedure in a specified period based on **parent_id**. The result can be queried only in the system database but cannot be queried in the user database.

  Return type: record

Parameter	Type	Description
parent_id	bigint	Specifies the unique_sql_id of the statement calling a stored procedure.
start_timestamp	timestamp with time zone	Start point of the SQL start time range.
end_timestamp	timestamp with time zone	End point of the SQL start time range.

Example:

```
gaussdb=# CREATE TABLE test(a int,b int);
CREATE TABLE
gaussdb=# INSERT INTO test values(1,1);
INSERT 0 1
```



```
gaussdb=# CREATE PROCEDURE mypro1() as num int;
gaussdb$# begin
gaussdb$# INSERT INTO test values(2,2);
gaussdb$# DELETE FROM test where a = 2;
gaussdb$# end;
gaussdb$# /
CREATE PROCEDURE

-- Enable the parameter to trace the substatements of the stored procedure.
gaussdb=# SET instr_unique_sql_track_type = 'all';
SET

-- Enable the parameter. Full statement records are generated in the dbe_perf.statement_history
table.
gaussdb=# SET track_stmt_stat_level = 'L0,L0';
SET

gaussdb=# CALL mypro1();
 mypro1
--------

(1 row)

gaussdb=# SET track_stmt_stat_level = 'off,L0';
SET

gaussdb=# SET instr_unique_sql_track_type = 'top';
SET

-- Query key information, which is used as a function parameter.
gaussdb=# SELECT query,unique_query_id,start_time,finish_time FROM dbe_perf.statement_history;
                query                 | unique_query_id |          start_time           |          finish_time
--------------------------------------+-----------------+-------------------------------
+-------------------------------
 set track_stmt_stat_level = 'L0,L0'; |       636388010 | 2023-06-02 17:40:49.176155+08 | 2023-06-02
17:40:49.176543+08
 call mypro1();                       |       536458473 | 2023-06-02 17:40:59.028144+08 | 2023-06-02
17:40:59.032027+08
 delete from test where a = ?         |       583323884 | 2023-06-02 17:40:59.029955+08 | 2023-06-02
17:40:59.031577+08
 insert into test values(?,?)         |       769279931 | 2023-06-02 17:40:59.029219+08 | 2023-06-02
17:40:59.029947+08
(4 rows)

-- Use unique_query_id, start time, and end time of the outer query statement as parameters to query
information about the specified stored procedure and its substatements within the period.
gaussdb=# SELECT query FROM
dbe_perf.get_full_sql_by_parent_id_and_timestamp(536458473,'2023-06-02
17:40:59.028144+08','2023-06-02 17:40:59.032027+08');
           query
-------------------------------
 call mypro1();
 delete from test where a = ?
 insert into test values(?,?)
(3 rows)

gaussdb=# DROP PROCEDURE mypro1();
DROP PROCEDURE
gaussdb=# DROP TABLE test;
DROP TABLE
```

7.6.26.14 Undo System Functions

To check the storage mode used by the current rollback segment, query the **undostoragetype** column in the gs_global_config system catalog. **segpage** indicates the segment-page mode, which is a reserved parameter and is not supported currently. **page** indicates the page mode.

Example:

```
gaussdb=# SELECT * FROM gs_global_config WHERE name LIKE '%undostoragetype%';
      name       |  value
-----------------+---------
 undostoragetype | page
(1 row)
```

- gs_undo_meta(type, zoneId, location)

  Description: Specifies metadata of a module in the undo system.

  Parameter description:

  – **type** (metadata type)

    - **0**: indicates metadata corresponding to an undo zone (record).
    - **1**: indicates metadata corresponding to an undo zone(transaction slot).
    - **2**: indicates metadata corresponding to an undo space (record).
    - **3**: indicates metadata corresponding to an undo space(transaction slot).

  – **zoneId** (undo zone ID)

    - **–1**: indicates metadata of all undo zones.
    - **0** to 1024 x 1024 – 1: indicates the metadata of the corresponding zone ID.

  – **location** (read location)

    - **0**: indicates that data is read from the current memory.
    - **1**: indicates that data is read from a physical file.

  Return type: record

  NOTE

  This system function supports only the rollback segments in page storage mode.

Table 7-113 gs_undo_meta parameters

Category	Parameter	Type	Description
Output parameter	zoneId	oid	Undo zone ID.
Output parameter	persistType	oid	Persistence level.
Output parameter	insert	text	Position of the next undo record to be inserted

Category	Parameter	Type	Description
Output parameter	discard	text	Position of the undo record that is recycled in common mode
Output parameter	end	text	Position of the undo record that is forcibly recycled. Values smaller than the value of this parameter indicate that the record has been recycled.
Output parameter	used	text	Undo space that has been used.
Output parameter	lsn	text	Modifies the LSN of an undo zone.
Output parameter	pid	oid	ID of a thread bound to an undo zone.

- gs_undo_translot(location, zoneId)

  Description: Specifies transaction slot information of the undo system.

  Parameter description:

  – **location** (read location)

    ▪ **0**: indicates that data is read from the current memory.

    ▪ **1**: indicates that data is read from a physical file.

  – **zoneId** (undo zone ID)

    ▪ **–1**: indicates metadata of all undo zones.

    ▪ **0** to 1024 x 1024 – 1: indicates the metadata of the corresponding zone ID.

  Return type: record

  NOTE

  This system function supports only the rollback segments in page storage mode.

Table 7-114 gs_undo_translot parameters

Category	Parameter	Type	Description
Output parameter	groupId	oid	Undo zone ID that is used.

Category	Parameter	Type	Description
Output parameter	xactId	text	Transaction ID.
Output parameter	startUndoPtr	text	Position where an undo record is inserted at the start of a transaction corresponding to a transaction slot.
Output parameter	endUndoPtr	text	Position where an undo record is inserted at the end of a transaction corresponding to a transaction slot.
Output parameter	lsn	text	Transaction slot pointer.
Output parameter	slot_states	oid	Transaction status. • **0** indicates that the transaction has been committed. • **1** indicates that the task is being executed. • **2** indicates that the rollback is in progress. • **3** indicates that the rollback is complete.

- gs_stat_undo([bool init])

  Description: Collects undo statistics.

  Return type: record

  NOTE

  When rollback segments are stored in files in page storage mode, the following format is used.

  Table 7-115 gs_stat_undo parameters

Category	Parameter	Type	Description
Input parameter	init	bool	(Optional) Specifies whether to clear statistics and restart statistics collection.
Output parameter	curr_used_zone_count	int	Number of used undo zones.

Category	Parameter	Type	Description
Output parameter	top_used_zones	text	Information about the first three undo zones with the maximum usage. The output format is as follows: • **zoneId1**: used size of zone 1. • **zoneId2**: used size of zone 2. • **zoneId3**: used size of zone 3.
Output parameter	curr_used_undo_size	int	Total size of the undo tablespace that is being used. The unit is MB.
Output parameter	undo_threshold	int	Calculation result of the value of the GUC parameter **undo_space_limit_size** x 80%. The unit is MB.
Output parameter	global_recycle_xid	xid	XID of the transaction recycled to the undo space. The undo records generated by the transaction whose XID is smaller than the value of XID are recycled.
Output parameter	oldest_xmin	xid	Oldest active transaction.
Output parameter	total_undo_chain_len	int8	Total length of all accessed undo chains.
Output parameter	max_undo_chain_len	int8	Maximum length of the accessed undo chain.
Output parameter	create_undo_file_count	uint32	Number of created undo files.
Output parameter	discard_undo_file_count	uint32	Number of deleted undo files.

Category	Parameter	Type	Description
Output parameter	info	text	If the input parameter is **false**, suggestions on the **undo_space_limit_size**, **undo_limit_size_per_transaction**, and **undo_retention_time** parameters are provided. If the input parameter is **true**, init (clearing statistics) is required, and only 'The statistics have been initialized.' is displayed.

Example 1: Clear undo statistics.

```
gaussdb=# SELECT * FROM gs_stat_undo(true);
 curr_used_zone_count |   top_used_zones    | curr_used_undo_size | undo_threshold |
global_recycle_xid | oldest_xmin | total_undo_chain_len | max_undo_chain_len | create_undo_file_coun
t | discard_undo_file_count |                info
----------------------+---------------------+---------------------+----------------+--------------------
+-------------+---------------------+--------------------+----------------------
--+-------------------------+--------------------------------------
                   3 | 0 : 0, 0 : 0, 0 : 0 |                   1 |         209715 |             15741 |       15741
|                  0 |                   0 |
2 |                     0 | The statistics have been initialized.
(1 row)
```

Example 2: Output undo statistics.

```
gaussdb=# SELECT * FROM gs_stat_undo(false);
 curr_used_zone_count |   top_used_zones    | curr_used_undo_size | undo_threshold |
global_recycle_xid | oldest_xmin | total_undo_chain_len | max_undo_chain_len | create_undo_file_coun
t | discard_undo_file_count |
            info

----------------------+---------------------+---------------------+----------------+--------------------
+-------------+---------------------+--------------------+----------------------
--+-------------------------
+-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------
-
                   3 | 0 : 0, 0 : 0, 0 : 0 |                   1 |         209715 |             16253 |       16253
|                  0 |                   0 |
2 |                     0 | Based on the statistic info from last initialization, undo_space_limit_size is
recommended to set >= 120953 blocks, undo_limit_size_per_transaction is
recommended to set <= 327150 blocks, undo_retention_time is recommended to set <= 259200
seconds. Since last initialization, max undo space used by a single transaction is 32715 blocks.
(1 row)
```

- gs_undo_record(undoptr)

  Description: Parses undo records.

  Parameter description:

  – **undoptr** (undo record pointer)

  Return type: record

  NOTE

  This system function supports only the rollback segments in page storage mode.

- gs_undo_dump_parsepage_mv(relpath text, blkno bigint, reltype text, rmem boolean)

  Description: Parses the page header information of the disk page in the Ustore table, header information of each tuple, flag bit information, and all historical undo version information that can be queried.

  Return type: text

  Note: Only the system administrator or O&M administrator can execute this function.

  NOTE

  Currently, this API supports only Ustore tables.

  Table 7-116 gs_undo_dump_parsepage_mv parameters

Category	Parameter	Type	Description
Input parameter	relpath	text	Relative path of the Ustore table data file, in the format of *Tablespace name*/*Database OID*/*relfilenode*. For example, **base/16603/16384**. You can run the **pg_relation_filepath('tablename')** command to query the relative path.
Input parameter	blkno	int8	• **–1**: All block pages are parsed. • **0**–*MaxBlocNumber*: A specified block page is parsed.
Input parameter	reltype	text	Table type. Currently, only the Ustore table is supported. The value is **uheap**.
Input parameter	rmem	Boolean	• **false** • **true** Currently, the value can only be **false**, indicating that the corresponding page is parsed from the disk file.
Output parameter	output	text	Absolute path of the parsing result file.

- gs_undo_meta_dump_zone(zone_id int, read_memory boolean)

  Description: Parses undo zone metadata in an undo module.

  Return type: record

  NOTE

  When rollback segments are stored in files in page storage mode, the following format is used.

Table 7-117 gs_undo_meta_dump_zone parameters

Category	Parameter	Type	Description
Input parameter	zone_id	int	Undo zone ID. • **–1**: All undo zones are queried. • **0**–**1048575**: The undo zone metadata is queried based on the zone ID.
Input parameter	read_memory	Boolean	• **true**: Data is read from the current memory. • **false**: Data is read from the physical file.
Output parameter	zone_id	oid	Undo zone ID.
Output parameter	persist_type	oid	Persistence levels: • **0**: ordinary table • **1**: unlogged table • **2**: temporary table
Output parameter	insert	text	Position of the next undo record to be inserted.
Output parameter	discard	text	Position of the undo record that is recycled in common mode.
Output parameter	forcediscard	text	Position of the undo record that is forcibly recycled. Values smaller than the value of this parameter indicate that the record has been recycled.
Output parameter	lsn	text	Modifies the LSN of an undo zone.

Example:

```
gaussdb=# SELECT * FROM gs_undo_meta_dump_zone(-1,true);
 zone_id | persist_type | insert | discard | forcediscard |   lsn
---------+--------------+--------+---------+--------------+----------
       0 |            0 | 244577 | 244577  | 244577       | 43967224
       1 |            0 | 108    | 66      | 66           | 43967568
  349525 |            1 | 24     | 24      | 24           | 0
  349526 |            1 | 24     | 24      | 24           | 0
```



```
 699050 |          2 | 24    | 24    | 24          | 0
 699051 |          2 | 24    | 24    | 24          | 0
(6 rows)
```

- gs_undo_meta_dump_spaces(zone_id int, read_memory boolean)

  Description: Parses metadata of an undo record space and a transaction slot space in an undo module.

  Return type: record

  NOTE

  This system function supports only the rollback segments in page storage mode.

  Table 7-118 gs_undo_meta_dump_spaces parameters

Category	Parameter	Type	Description
Input parameter	zone_id	int	Undo zone ID. • **–1**: All undo zones are queried. • **0**–**1048575**: The undo zone metadata is queried based on the zone ID.
Input parameter	read_memory	Boolean	• **true**: Data is read from the current memory. • **false**: Data is read from the physical file.
Output parameter	zone_id	int	Undo zone ID.
Output parameter	undorecord_space_tail	text	End position of the undo record space
Output parameter	undorecord_space_head	text	Start position of the undo record space
Output parameter	undorecord_space_lsn	text	Modifies the LSN of an undo record space.
Output parameter	undoslot_space_tail	text	End position of a transaction slot space

Category	Parameter	Type	Description
Output parameter	undoslot_space_head	text	Start position of a transaction slot space
Output parameter	undoreslot_space_lsn	text	Modifies the LSN of a transaction slot space.

- gs_undo_meta_dump_slot(zone_id int, read_memory boolean)
  Description: Parses transaction slot metadata in an undo module.
  Return type: record

  NOTE

  This system function supports only the rollback segments in page storage mode.

  Table 7-119 gs_undo_meta_dump_slot parameters

Category	Parameter	Type	Description
Input parameter	zone_id	int	Undo zone ID. • **–1**: All undo zones are queried. • **0–1048575**: The undo zone metadata is queried based on the zone ID.
Input parameter	read_memory	Boolean	• **true**: Data is read from the current memory. • **false**: Data is read from the physical file.
Output parameter	zone_id	int	Undo zone ID.
Output parameter	allocate	text	Allocation position of the undo transaction slot.
Output parameter	recycle	text	Recycling position of the undo transaction slot.

Category	Parameter	Type	Description
Output parameter	frozen_xid	text	Frozen XID, which is used to determine the visibility.
Output parameter	global_frozen_xid	text	Minimum frozen XID in the system. Transactions whose XID is smaller than the value of this parameter are visible.
Output parameter	recycle_xid	text	Recycled XID. Transactions whose XID is smaller than the value of this parameter are recycled.
Output parameter	global_recycle_xid	text	Minimum recycled XID in the system. Transactions whose XID is smaller than the value of this parameter are recycled.

- gs_undo_translot_dump_slot(zone_id int, read_memory boolean)

  Description: Parses a transaction slot in an undo zone.

  Return type: record

  NOTE

  When rollback segments are stored in files in page storage mode, the following format is used.

  Table 7-120 gs_undo_translot_dump_slot parameters

Category	Parameter	Type	Description
Input parameter	zone_id	int	Undo zone ID. • **–1**: All undo zones are queried. • **0**–**1048575**: The undo zone metadata is queried based on the zone ID.
Input parameter	read_memory	Boolean	• **true**: Data is read from the current memory. • **false**: Data is read from the physical file.
Output parameter	zone_id	oid	Undo zone ID.
Output parameter	slot_xid	text	Transaction ID.

Category	Parameter	Type	Description
Output parameter	start_undoptr	text	Position where the undo record is inserted at the start of the transaction corresponding to a slot
Output parameter	end_undoptr	text	Position where the undo record is inserted at the end of the transaction corresponding to a slot
Output parameter	slot_ptr	text	Position of a transaction slot.
Output parameter	slot_states	oid	Transaction state • **0**: committed • **1**: being executed • **2**: being rolled back • **3**: rolled back

Example:

```
gaussdb=# SELECT * FROM gs_undo_translot_dump_slot(-1,true);
 zone_id |     slot_xid     |  start_undoptr   |   end_undoptr    |     slot_ptr     | slot_states
---------+------------------+------------------+------------------+------------------+-------------
       1 | 0000000000015758 | 0000000000000042 | 000000000000006C | 0000000000000038 |           0
(1 row)
```

- gs_undo_translot_dump_xid(slot_xid xid, read_memory boolean)

  Description: Parses a transaction slot in an undo zone based on the XID.

  Return type: record

  NOTE

  When rollback segments are stored in files in page storage mode, the following format is used.

Table 7-121 gs_undo_translot_dump_xid parameters

Category	Parameter	Type	Description
Input parameter	slot_xid	xid	Transaction ID to be queried.
Input parameter	read_memory	Boolean	• **true**: Data is read from the current memory. • **false**: Data is read from the physical file.

Category	Parameter	Type	Description
Output parameter	zone_id	oid	Undo zone ID.
Output parameter	slot_xid	text	Transaction ID.
Output parameter	start_undoptr	text	Position where the undo record is inserted at the start of the transaction corresponding to a slot
Output parameter	end_undoptr	text	Position where the undo record is inserted at the end of the transaction corresponding to a slot.
Output parameter	slot_ptr	text	Position of a transaction slot.
Output parameter	slot_states	oid	Transaction state. • **0**: committed • **1**: being executed • **2**: being rolled back • **3**: rolled back

Example:

```
gaussdb=# SELECT * FROM gs_undo_translot_dump_xid('15758',false);
 zone_id |     slot_xid     |  start_undoptr   |   end_undoptr    |     slot_ptr     | slot_states
---------+------------------+------------------+------------------+------------------+-------------
       1 | 0000000000015758 | 0000000000000042 | 000000000000006C | 0000000000000038 |           0
(1 row)
```

- gs_undo_dump_record(undoptr bigint)

  Description: Parses the undo record information of a specified URP.

  Return type: record

  NOTE

  When rollback segments are stored in files in page storage mode, the following format is used.

Table 7-122 gs_undo_dump_record parameters

Category	Parameter	Type	Description
Input parameter	undoptr	xid	Start position of the undo record to be parsed.

Category	Parameter	Type	Description
Output parameter	undoptr	xid	Start position of the undo record to be parsed.
Output parameter	xactid	xid	Transaction ID.
Output parameter	cid	text	Command ID.
Output parameter	reloid	text	Relation OID.
Output parameter	relfilenode	text	Relfinode of the file.
Output parameter	utype	text	Undo record type.
Output parameter	blkprev	text	Position of the previous undo record in the same block.
Output parameter	blockno	text	Block number.
Output parameter	uoffset	text	Undo record offset.
Output parameter	prevurp	text	Position of the previous undo record.
Output parameter	payloadlen	text	Length of the undo record data.
Output parameter	oldxactid	text	Previous transaction ID.
Output parameter	partitionoid	text	Partition OID.

Category	Parameter	Type	Description
Output parameter	tablespace	text	Tablespace.
Output parameter	alreadyread_bytes	text	Length of the read undo record.
Output parameter	prev_undorec_len	text	Length of the previous undo record.
Output parameter	td_id	text	ID of the transaction directory.
Output parameter	reserved	text	Reserved flag of the old tuple stored in an undo record.
Output parameter	flag	text	Status flag of the old tuple stored in an undo record.
Output parameter	flag2	text	Number of old tuple columns stored in an undo record.
Output parameter	t_hoff	text	Length of the undo record data header

Example (If the undo record is not recycled, the input parameter undoptr can be queried after being converted to a decimal value using the output parameter **end_undoptr** of the gs_undo_translot_dump_slot function.):

```
gaussdb=# SELECT * FROM gs_undo_dump_record('0000000000000042');
 undoptr |   xactid   | cid | reloid | relfilenode | utype |  blkprev   | blockno | uoffset | prevurp |
payloadlen | oldxactid | partitionoid | tablespace | alreadyread_byt
es | prev_undorec_len | td_id | reserved | flag | flag2 | t_hoff
---------+------------+-----+--------+-------------+-------+------------+---------+---------+---------
+------------+-----------+--------------+------------+----------------
---+------------------+-------+----------+------+-------+--------
     42 | 1073807360 | 0   | 0      | 108986369   | 0     | 1024786474 | 0       | 0       | 0       |
0          | 0           | 0          | 36
  | 16390           | -1    | -1       | -1   | -1    | -1
(1 row)
```

- gs_undo_dump_xid(undo_xid xid)

  Description: Parses undo records based on the XID.

  Return type: record

NOTE

When rollback segments are stored in files in page storage mode, the following format is used.

Table 7-123 gs_undo_dump_xid parameters

Category	Parameter	Type	Description
Input parameter	undo_xid	xid	Transaction XID.
Output parameter	undoptr	xid	Start position of the undo record to be parsed.
Output parameter	xactid	xid	Transaction ID.
Output parameter	cid	text	Command ID.
Output parameter	reloid	text	Relation OID.
Output parameter	relfilenode	text	Relfinode of the file.
Output parameter	utype	text	Undo record type.
Output parameter	blkprev	text	Position of the previous undo record in the same block.
Output parameter	blockno	text	Block number.
Output parameter	uoffset	text	Undo record offset.
Output parameter	prevurp	text	Position of the previous undo record.

Category	Parameter	Type	Description
Output parameter	payloadlen	text	Length of the undo record data.
Output parameter	oldxactid	text	Previous transaction ID.
Output parameter	partitionoid	text	Partition OID.
Output parameter	tablespace	text	Tablespace.
Output parameter	alreadyread_bytes	text	Length of the read undo record.
Output parameter	prev_undorec_len	text	Length of the previous undo record.
Output parameter	td_id	text	ID of the transaction directory.
Output parameter	reserved	text	Reserved flag of the old tuple stored in an undo record.
Output parameter	flag	text	Status flag of the old tuple stored in an undo record.
Output parameter	flag2	text	Number of old tuple columns stored in an undo record.
Output parameter	t_hoff	text	Length of the undo record data header

Example:

```
gaussdb=# SELECT * FROM gs_undo_dump_xid('15779');
 undoptr | xactid | cid | reloid | relfilenode | utype | blkprev | blockno | uoffset | prevurp | payloadlen |
oldxactid | partitionoid | tablespace | alreadyread_bytes | pr
ev_undorec_len | td_id | reserved | flag | flag2 | t_hoff
---------+--------+-----+--------+-------------+-------+---------+---------+---------+---------+------------
+-----------+--------------+------------+-------------------+---
---------------+-------+----------+------+-------+--------
   108 |  15779 | 0   | 16767  | 16767       | 64    | 0       | 0       | 3       | 0       | 0         | 0        |
```



```
0          | 1663       | 40              | 42
           | -1    | -1       | -1   | -1    | -1
(1 row)
```

- gs_verify_undo_record(type, start_idx, end_idx, location)

  Description: Verifies undo records. Currently, only the disk verification mode is supported. Offline verification can be performed only when services are not running. Before the verification, you need to manually perform checkpoint flushing.

  Return type: record

  NOTE

  When rollback segments are stored in files in page storage mode, the following format is used.

Table 7-124 gs_verify_undo_record parameters

Category	Parameter	Type	Description
Input parameter	type	text	Verification type. ● **'urp'**: verifies all undo records in a specified URP range. ● **'zone'**: verifies all undo records of all zones in a specified zone range.
Input parameter	start_idx	int8	Start position. ● When **type** is set to **'urp'**, this parameter indicates the start position of undo records. ● When type is set to **'zone'**, this parameter indicates the start undo zone ID.
Input parameter	end_idx	int8	End position. ● When **type** is set to **'urp'**, this parameter indicates the end position of undo records. ● When **type** is set to **'zone'**, this parameter indicates the end undo zone ID.
Input parameter	location	bool	● **0**: memory verification ● **1**: disk verification Currently, this parameter can only be set to **1**.
Output parameter	zone_id	int8	Undo zone ID.

Category	Parameter	Type	Description
Output parameter	detail	text	Verification error information.

Example 1: Verify the undo record whose URP is 24.

```
gaussdb=# SELECT * FROM gs_verify_undo_record('urp', 24, 24, 1);
zone_id | detail
--------+--------
(0 rows)
```

Example 2: Verify all undo records from zone 0 to zone 2 on the disk.

```
gaussdb=# SELECT * FROM gs_verify_undo_record('zone', 0, 2, 1);
zone_id | detail
--------+--------
(0 rows)
```

NOTE

If an error is reported when this view is called, contact Huawei technical support.

- gs_verify_undo_slot(type, start_idx, end_idx, location)

  Description: Verifies undo transaction slots. Currently, only the disk verification mode is supported. Offline verification can be performed only when services are not running. Before the verification, you need to manually perform checkpoint flushing.

  Return type: record

  NOTE

  When rollback segments are stored in files in page storage mode, the following format is used.

Table 7-125 gs_verify_undo_slot parameters

Category	Parameter	Type	Description
Input parameter	type	text	Verification type. • **'zone'**: verifies all transaction slots of all zones in a specified zone range.
Input parameter	start_idx	int8	Start undo zone ID.
Input parameter	end_idx	int8	End undo zone ID.

Category	Parameter	Type	Description
Input parameter	location	bool	• **0**: memory verification • **1**: disk verification Currently, this parameter can only be set to **1**.
Output parameter	zone_id	int8	Undo zone ID.
Output parameter	detail	text	Verification error information.

Example: Verify all transaction slot records from zone 0 to zone 2 on the disk.

```
gaussdb=# select * from gs_verify_undo_slot('zone', 0, 2, 1);
zone_id | detail
--------+--------
(0 rows)
```

NOTE

If an error is reported when this view is called, contact Huawei technical support.

- gs_verify_undo_meta(type, start_idx, end_idx, location)

  Description: Verifies undo metadata. Currently, only the disk verification mode is supported. Offline verification can be performed only when services are not running. Before the verification, you need to manually perform checkpoint flushing.

  Return type: record

  NOTE

  This system function supports only the rollback segments in page storage mode.

Table 7-126 gs_verify_undo_meta parameters

Category	Parameter	Type	Description
Input parameter	type	text	Verification type. The value of **type** can only be set to **'all'**. • **'all'**: verifies all meta information of all zones in a specified zone range.
Input parameter	start_idx	int64	Start undo zone ID.
Input parameter	end_idx	int64	End undo zone ID.

Category	Parameter	Type	Description
Input parameter	location	bool	• **0**: memory verification • **1**: disk verification Currently, this parameter can only be set to **1**.
Output parameter	zone_id	int64	Undo zone ID.
Output parameter	detail	text	Verification error information.

Example: Verify all meta information records from zone 0 to zone 2 on the disk.

```
gaussdb=# SELECT * FROM gs_verify_undo_meta('all', 0, 2, 1);
zone_id | detail
--------+--------
(0 rows)
```

NOTE

If an error is reported when this view is called, contact Huawei technical support.

- gs_async_rollback_worker_status()

  Description: Monitors the status of active asynchronous rollback threads.

  Return type: record

  Table 7-127 gs_async_rollback_worker_status parameters

Category	Parameter	Type	Description
Output parameter	datid	oid	Database ID.
Output parameter	pid	int8	Thread ID.
Output parameter	sessionid	int8	Session ID.
Output parameter	usesysid	oid	ID of the user who initiates the thread.
Output parameter	state	int	Thread status. **0**: undefined **1**: idle **2**: running

Category	Parameter	Type	Description
Output parameter	rollback_start_time	timestamptz	Timestamp when a thread is started.
Output parameter	idx	oid	Index of an asynchronous rollback thread in the array.
Output parameter	xid	xid	XID of the transaction that is being rolled back.
Output parameter	progress	text	Rollback progress of the transaction (number of undo records that have been rolled back/total number of undo records, in percentage).

- gs_async_rollback_xact_status()
  Description: Monitors the hash table of asynchronous rollback tasks.
  Return type: record

  Table 7-128 gs_async_rollback_xact_status parameters

Category	Parameter	Type	Description
Output parameter	xid	xid	XID of the transaction that requires asynchronous rollback.
Output parameter	start_undoptr	xid	Pointer to the start undo record of the transaction.
Output parameter	end_undoptr	xid	Pointer to the end undo record of the transaction.
Output parameter	dbid	uint32	ID of the database where the transaction is located.
Output parameter	slot_ptr	xid	Pointer to the transaction slot corresponding to the transaction.
Output parameter	launched	bool	Checks whether there is an active asynchronous rollback thread.

- gs_undo_recycler_status()
  Description: Monitors the status of asynchronous recycling threads.
  Return type: record

Table 7-129 gs_undo_recycler_status parameters

Category	Parameter	Type	Description
Output parameter	datid	oid	Database ID.
Output parameter	pid	int8	Thread ID.
Output parameter	sessionid	int8	Session ID.
Output parameter	usesysid	oid	ID of the user who initiates the thread.
Output parameter	state	int	Thread status. **0**: undefined **1**: idle **2**: running
Output parameter	backend_start	timestamptz	Timestamp when a thread is started.
Output parameter	total_recycle_time	xid	Total recycling time.
Output parameter	max_recycle_time	xid	Maximum recycling time.
Output parameter	total_recycle_size	xid	Total recycled space.
Output parameter	total_recycle_count	xid	Total number of recycling times.
Output parameter	recycle_sleep_count	xid	Number of sleep times.
Output parameter	recycle_sleep_time	xid	Total sleep time.
Output parameter	max_recycle_sleep_time	xid	Maximum sleep time.
Output parameter	last_recycle_timestamp	timestamptz	Timestamp of the last successful recycling.
Output parameter	last_update_global_recycle_xid_timestamp	timestamptz	Timestamp of the last global recycling transaction.

- gs_undo_launcher_status()

  Description: Monitors the status of asynchronous rollback launcher threads.

  Return type: record

Table 7-130 gs_undo_launcher_status parameters

Category	Parameter	Type	Description
Output parameter	datid	oid	Database ID.
Output parameter	pid	int64	Thread ID.
Output parameter	sessionid	int64	Session ID.
Output parameter	usesysid	oid	ID of the user who initiates the thread.
Output parameter	state	int32	Thread status. **0**: undefined **1**: idle **2**: running
Output parameter	backend_start	timestamptz	Timestamp when a thread is started.
Output parameter	total_async_rollback_task_count	uint64	Total number of asynchronous rollback tasks initiated after the database on the local node is started.
Output parameter	average_async_rollback_time	uint64	Average duration of an asynchronous rollback task.
Output parameter	max_async_rollback_time	uint64	Maximum duration of an asynchronous rollback task.
Output parameter	min_async_rollback_time	uint64	Minimum duration of an asynchronous rollback task.

7.6.27 SPM Functions

- GS_SPM_EVOLUTE_PLAN(sql_hash, plan_hash)

  Description: GS_SPM_EVOLUTE_PLAN belongs to the DBE_SQL_UTIL schema and is a function used by the SPM feature to plan evolution. As long as the table related to the plan baseline exists, the plan baseline can be evolved.

  Parameters: See **Table 7-131**.

Table 7-131 GS_SPM_EVOLUTE_PLAN input parameters and return values

Parameter	Type	Description	Value Range
sql_hash	IN bigint	Specifies a hash value of SQL text.	-
plan_hash	IN bigint	Specifies a hash value of the outline text of the SQL plan.	-
evolute_status	OUT boolean	Determines whether the evolution is complete. **t** indicates normal. If an exception occurs, an error is reported.	t/f

Return type: Boolean

Example:

```
gaussdb=# SELECT * FROM dbe_sql_util.gs_spm_evolute_plan(107760189, 2284373089);
 evolute_status
----------------
 t
(1 row)
```

- GS_SPM_SET_PLAN_STATUS(sql_hash, plan_hash, plan_status)

  Description: GS_SPM_SET_PLAN_STATUS belongs to the DBE_SQL_UTIL schema and is a function used by the SPM feature to change the baseline status.

  Parameters: See **Table 7-132**.

  Table 7-132 GS_SPM_SET_PLAN_STATUS input parameters and return values

Parameter	Type	Description	Value Range
sql_hash	IN bigint	Specifies a hash value of SQL text.	-
plan_hash	IN bigint	Specifies a hash value of the outline text of the SQL plan.	-

Parameter	Type	Description	Value Range
plan_status	IN text	Specifies the status of a plan.	• **ACC**: indicates that the plan has been accepted. • **UNACC**: indicates that the plan is not accepted. • **FIXED**: indicates a special ACC plan. The matching priority of this plan is higher than that of other ACC plans.
execute_status	OUT boolean	Determines whether the plan status change is complete. **t** indicates normal. If an exception occurs, an error is reported.	t/f

Return type: Boolean

Example:

```
gaussdb=# SELECT dbe_sql_util.gs_spm_set_plan_status(sql_hash, plan_hash, 'ACC') FROM
gs_spm_sql_baseline
where outline like '%BitmapScan%';
 gs_spm_set_plan_status
------------------------
 t
(1 row)
```

- GS_SPM_DISPLAY_PLANS(sql_hash)

  Description: GS_SPM_DISPLAY_PLANS belongs to the DBE_SQL_UTIL schema and is a function used by the SPM feature to view all baselines of a single SQL statement.

  Parameters: See **Table 7-133**.

  Table 7-133 GS_SPM_DISPLAY_PLANS input parameters and return values

Parameter	Type	Description	Value Range
sql_hash	IN bigint	Unique ID of an SQL statement	-

Parameter	Type	Description	Value Range
sql_hash	OUT bigint	Specifies a hash value of SQL text.	-
plan_hash	OUT bigint	Specifies a hash value of the outline text of the SQL plan.	-
outline	OUT text	Specifies the combination character string of all hints of the outline corresponding to the current plan.	-
cost	OUT double	Specifies the cost of the current plan.	-
status	OUT text	Specifies the status of the current plan.	• **ACC**: indicates that the plan has been accepted. • **UNACC**: indicates that the plan is not accepted. • **FIXED**: indicates a special ACC plan. The matching priority of this plan is higher than that of other ACC plans.
gplan	OUT boolean	Determines whether the current plan is a gplan. **t** indicates gplan. If an exception occurs, an error is reported.	t/f

Return type: bigint, text, double, text, boolean

Example:

```
select sql_hash, plan_hash, outline, status, gplan from dbe_sql_util.gs_spm_display_plans(107760189)
order by status, outline;
 sql_hash  |  plan_hash   |                   outline                    | status | gplan
-----------+------------+----------------------------------------------+--------+-------
 107760189 | 2519317082 |  begin_outline_data                          +| ACC    | f
           |            |  IndexScan(@"sel$1" tb_a@"sel$1" tb_a_idx_c1) +|        |
           |            |  version("1.0.0")                            +|        |
           |            |  end_outline_data                            |        |
 107760189 | 2686653876 |  begin_outline_data                          +| UNACC  | f
           |            |  BitmapScan(@"sel$1" tb_a@"sel$1" tb_a_idx_c1)+|       |
           |            |  version("1.0.0")                            +|        |
           |            |  end_outline_data                            |        |
 107760189 | 2284373089 |  begin_outline_data                          +| UNACC  | f
           |            |  TableScan(@"sel$1" tb_a@"sel$1")             +|        |
           |            |  version("1.0.0")                            +|        |
```

```
        |          | end_outline_data                        |      |
(3 rows)
```

- GS_SPM_RELOAD_PLAN(sql_hash, plan_hash)

  Description: GS_SPM_RELOAD_PLAN belongs to the DBE_SQL_UTIL schema and is a function used by the SPM feature to load a baseline in the baseline system catalog to the SPM global cache.

  Parameters: See **Table 7-134**.

  Table 7-134 GS_SPM_RELOAD_PLAN input parameters and return values

Parameter	Type	Description	Value Range
sql_hash	IN bigint	Specifies a hash value of SQL text.	-
plan_hash	IN bigint	Specifies a hash value of the outline text of the SQL plan.	-
execute_status	OUT boolean	Determines whether the baseline loading is complete. **t** indicates normal. If an exception occurs, an error is reported.	t/f

  Return type: Boolean

  Example:

```
SELECT dbe_sql_util.gs_spm_reload_plan(sql_hash, plan_hash) from gs_spm_sql_baseline where
outline like '%IndexScan%';
 gs_spm_reload_plan
--------------------
 t
(1 row)
```

- GS_SPM_VALIDATE_PLAN(sql_hash, plan_hash)

  Description: GS_SPM_VALIDATE_PLAN belongs to the DBE_SQL_UTIL schema and is a function used by the SPM feature to verify plan availability.

  Parameters: See **Table 7-135**.

  Table 7-135 GS_SPM_VALIDATE_PLAN input parameters and return values

Parameter	Type	Description	Value Range
sql_hash	IN bigint	Specifies a hash value of SQL text.	-
plan_hash	IN bigint	Specifies a hash value of the outline text of the SQL plan.	-
execute_status	OUT boolean	Determines whether the verified plan is available: **t**: The verified plan is available. **f**: The verified plan is unavailable.	t/f

Return type: Boolean

Example:

```
SELECT dbe_sql_util.gs_spm_validate_plan(sql_hash, plan_hash) FROM gs_spm_sql_baseline WHERE
outline LIKE '%IndexScan%';
 gs_spm_validate_plan
----------------------
 f
(1 row)
```

- GS_SPM_DELETE_PLAN(sql_hash, plan_hash)

  Description: GS_SPM_DELETE_PLAN belongs to DBE_SQL_UTIL schema and is a function used by the SPM feature to delete the plan baseline. If the function is aborted during execution, the number of records in the gs_spm_baseline table may exceed the value of **spm_plan_capture_max_plannum**.

  Parameters: See **Table 7-136**.

  Table 7-136 GS_SPM_DELETE_PLAN input parameters and return values

Parameter	Type	Description	Value Range
sql_hash	IN bigint	Specifies a hash value of SQL text.	-
plan_hash	IN bigint	Specifies a hash value of the outline text of the SQL plan.	-
execute_status	OUT boolean	Determines whether the plan deletion is complete. **t** indicates normal. If an exception occurs, an error is reported.	t/f

  Return type: Boolean

  Example:

```
SELECT dbe_sql_util.gs_spm_delete_plan(sql_hash, plan_hash) FROM gs_spm_sql_baseline WHERE
outline LIKE '%IndexScan%';
 gs_spm_delete_plan
--------------------
 t
(1 row)
```

7.6.28 Statistics Information Functions

Statistics information functions are divided into the following two categories: functions that access databases, using the OID of each table or index in a database to mark the database for which statistics are generated; functions that access servers, identified by the server thread ID, whose value ranges from 1 to the number of currently active servers.

- pg_stat_get_db_conflict_tablespace(oid)

  Description: Specifies the number of queries canceled due to a conflict between the restored tablespace and the deleted tablespace in the database.

Return type: bigint

- pg_control_group_config()

  Description: Prints Cgroup configurations on the current node. Only users with the SYSADMIN permission can execute this function.

  Return type: record

- pg_stat_get_db_stat_reset_time(oid)

  Description: Specifies the most recent time when database statistics were reset. It is initialized to the system time during the first connection to each database. The reset time is updated when you call **pg_stat_reset** on the database and execute **pg_stat_reset_single_table_counters** against any table or index in it.

  Return type: timestamptz

- pg_stat_get_function_total_time(oid)

  Description: Specifies the total wall clock time spent in the function, in microseconds. The time spent on this function calling other functions is included.

  Return type: bigint

- pg_stat_get_xact_tuples_returned(oid)

  Description: Specifies the number of rows read through sequential scans when the parameter is a table in the current transaction or the number of index entries returned when the parameter is an index.

  Return type: bigint

- pg_lock_status()

  Description: Queries information about locks held by open transactions. All users can execute this function.

  Return type: For details, see **PG_LOCKS** which is obtained by querying this function.

- gs_lwlock_status()

  Description: Queries information about all lightweight locks in the database system, including lock waiting and lock holding information. All users can execute this function.

  Return type: setofrecord

- pg_stat_get_xact_numscans(oid)

  Description: Specifies the number of sequential scans performed when the parameter is a table in the current transaction or the number of index scans performed when the parameter is an index.

  Return type: bigint

- pg_stat_get_xact_blocks_fetched(oid)

  Description: Specifies the number of disk block fetch requests for a table or an index in the current transaction.

  Return type: bigint

- pg_stat_get_xact_blocks_hit(oid)

  Description: Specifies the number of disk block fetch requests for tables or indexes found in cache in the current transaction.

  Return type: bigint

- pg_stat_get_xact_function_calls(oid)

  Description: Specifies the number of times the function is called in the current transaction.

  Return type: bigint

- pg_stat_get_xact_function_self_time(oid)

  Description: Specifies the time spent on this function in the current transaction, excluding the time spent on this function internally calling other functions.

  Return type: bigint

- pg_stat_get_xact_function_total_time(oid)

  Description: Specifies the total wall clock time (in microseconds) spent on the function in the current transaction, including the time spent on this function internally calling other functions.

  Return type: bigint

- pg_stat_get_wal_senders()

  Description: Queries walsender information on the primary node.

  Return type: setofrecord

  The following table describes return columns.

Table 7-137 Return column description

Column	Type	Description
pid	bigint	Thread ID of the WAL sender
sender_pid	integer	Lightweight thread ID of the WAL sender
local_role	text	Type of the primary node
peer_role	text	Type of the standby node
peer_state	text	Status of the standby node
state	text	Status of the WAL sender
catchup_start	timestamp with time zone	Startup time of a catchup task
catchup_end	timestamp with time zone	End time of a catchup task
sender_sent_location	text	Sending position of the primary node
sender_write_location	text	Writing position of the primary node

Column	Type	Description
sender_flush_location	text	Flushing position of the primary node
sender_replay_location	text	Redo position of the primary node
receiver_received_location	text	Receiving position of the standby node
receiver_write_location	text	Writing position of the standby node
receiver_flush_location	text	Flushing position of the standby node
receiver_replay_location	text	Redo position of the standby node
sync_percent	text	Synchronization percentage
sync_state	text	Synchronization status
sync_group	text	Group to which the synchronous replication belongs
sync_priority	text	Priority of synchronous replication
sync_most_available	text	Maximum availability mode
channel	text	Channel information of the WAL sender

- get_paxos_replication_info()
  Description: Queries the primary/standby replication status in Paxos mode.
  Return type: setofrecord
  The following table describes return columns.

  Table 7-138 Return column description

Column	Type	Description
paxos_write_location	text	Location of the Xlog that has been written to the Distribute Consensus Framework (DCF)
paxos_commit_location	text	Location of the Xlog agreed in the DCF

Column	Type	Description
local_write_location	text	Writing position of a node
local_flush_location	text	Flushing position of a node
local_replay_location	text	Redo position of a node
dcf_replication_info	text	DCF module information of a node

- pg_stat_get_stream_replications()
  Description: Queries the primary/standby replication status.
  Return type: setofrecord
  The following table describes return values.

  Table 7-139 Return value description

Return Parameter	Type	Description
local_role	text	Local role
static_connections	integer	Connection statistics
db_state	text	Database status
detail_information	text	Detailed information

- pg_stat_get_db_numbackends(oid)
  Description: Specifies the number of active server threads for a database.
  Return type: integer
- pg_stat_get_db_xact_commit(oid)
  Description: Specifies the number of transactions committed in a database.
  Return type: bigint
- pg_stat_get_db_xact_rollback(oid)
  Description: Specifies the number of transactions rolled back in a database.
  Return type: bigint
- pg_stat_get_db_blocks_fetched(oid)
  Description: Specifies the number of disk blocks fetch requests for a database.
  Return type: bigint
- pg_stat_get_db_blocks_hit(oid)
  Description: Specifies the number of disk block fetch requests found in cache for a database.
  Return type: bigint

- pg_stat_get_db_tuples_returned(oid)

  Description: Specifies the number of tuples returned for a database.

  Return type: bigint
- pg_stat_get_db_tuples_fetched(oid)

  Description: Specifies the number of tuples fetched for a database.

  Return type: bigint
- pg_stat_get_db_tuples_inserted(oid)

  Description: Specifies the number of tuples inserted in a database.

  Return type: bigint
- pg_stat_get_db_tuples_updated(oid)

  Description: Specifies the number of tuples updated in a database.

  Return type: bigint
- pg_stat_get_db_tuples_deleted(oid)

  Description: Specifies the number of tuples deleted in a database.

  Return type: bigint
- pg_stat_get_db_conflict_lock(oid)

  Description: Specifies the number of lock conflicts in a database.

  Return type: bigint
- pg_stat_get_db_deadlocks(oid)

  Description: Specifies the number of deadlocks in a database.

  Return type: bigint
- pg_stat_get_numscans(oid)

  Description: Number of sequential row scans done if parameters are in a table, or the number of index rows if parameters are in an index.

  Return type: bigint
- pg_stat_get_role_name(oid)

  Description: Obtains the username based on the user OID. Only users with the SYSADMIN and MONADMIN permissions can access the information.

  Return type: text

  Example:

  ```
  gaussdb=# SELECT pg_stat_get_role_name(10);
   pg_stat_get_role_name
  -----------------------
   aabbcc
  (1 row)
  ```
- pg_stat_get_tuples_returned(oid)

  Description: Specifies the number of sequential rows read by sequential scans if parameters are in a table, or the number of index rows if parameters are in an index.

  Return type: bigint
- pg_stat_get_tuples_fetched(oid)

  Description: Specifies the number of table rows fetched by bitmap scans if parameters are in a table, or the number of table rows fetched by simple index scans in the original table if parameters are in an index.

Return type: bigint

- pg_stat_get_tuples_inserted(oid)

  Description: Specifies the number of rows inserted into a table.

  Return type: bigint

- pg_stat_get_tuples_updated(oid)

  Description: Specifies the number of rows updated in a table.

  Return type: bigint

- pg_stat_get_tuples_deleted(oid)

  Description: Specifies the number of rows deleted from a table.

  Return type: bigint

- pg_stat_get_tuples_changed(oid)

  Description: Specifies the total number of inserted, updated, and deleted rows after a table was last analyzed or autoanalyzed.

  Return type: bigint

- pg_stat_get_tuples_hot_updated(oid)

  Description: Specifies the number of rows hot updated in a table.

  Return type: bigint

- pg_stat_get_live_tuples(oid)

  Description: Specifies the number of live rows in a table.

  Return type: bigint

- pg_stat_get_dead_tuples(oid)

  Description: Specifies the number of dead rows in a table.

  Return type: bigint

- pg_stat_get_blocks_fetched(oid)

  Description: Specifies the number of disk block fetch requests for a table or an index.

  Return type: bigint

- pg_stat_get_blocks_hit(oid)

  Description: Specifies the number of disk block requests found in cache for a table or an index.

  Return type: bigint

- pg_stat_get_xact_tuples_fetched(oid)

  Description: Specifies the number of tuple rows scanned in a transaction.

  Return type: bigint

- pg_stat_get_xact_tuples_inserted(oid)

  Description: Specifies the number of tuple inserted into the active subtransactions related to a table.

  Return type: bigint

- pg_stat_get_xact_tuples_deleted(oid)

  Description: Specifies the number of deleted tuples in the active subtransactions related to a table.

  Return type: bigint

- pg_stat_get_xact_tuples_hot_updated(oid)

  Description: Specifies the number of hot updated tuples in the active subtransactions related to a table.

  Return type: bigint

- pg_stat_get_xact_tuples_updated(oid)

  Description: Specifies the number of updated tuples in the active subtransactions related to a table.

  Return type: bigint

- pg_stat_get_last_vacuum_time(oid)

  Description: Specifies the most recent time when the autovacuum thread is manually started to clear a table.

  Return type: timestamptz

- pg_stat_get_last_autovacuum_time(oid)

  Description: Specifies the time of the last vacuum initiated by the autovacuum daemon thread on a table.

  Return type: timestamptz

- pg_stat_get_vacuum_count(oid)

  Description: Specifies the number of times a table is manually cleared.

  Return type: bigint

- pg_stat_get_autovacuum_count(oid)

  Description: Specifies the number of times the autovacuum daemon thread is started to clear a table.

  Return type: bigint

- pg_stat_get_last_analyze_time(oid)

  Description: Specifies the last time when a table starts to be analyzed manually or by the autovacuum thread.

  Return type: timestamptz

- pg_stat_get_last_autoanalyze_time(oid)

  Description: Specifies the time when the last analysis initiated by the autovacuum daemon thread on a table.

  Return type: timestamptz

- pg_stat_get_analyze_count(oid)

  Description: Specifies the number of times a table is manually analyzed.

  Return type: bigint

- pg_stat_get_autoanalyze_count(oid)

  Description: Specifies the number of times the autovacuum daemon thread analyzes a table.

  Return type: bigint

- pg_total_autovac_tuples(bool)

  Description: Returns tuple records related to the total autovac, such as **nodename**, **nspname**, **relname**, and tuple IUDs. The input parameter specifies whether to query the **relation** information.

  Return type: setofrecord

Output parameters: See **Table 7-140**.

Table 7-140 Return parameters

Return Parameter	Type	Description
nodename	name	Node name.
nspname	name	Name of a namespace
relname	name	Name of an object, such as a table, an index, or a view
partname	name	Partition name
n_dead_tuples	bigint	Number of dead rows in a table partition
n_live_tuples	bigint	Number of live rows in a table partition
changes_since_analyze	bigint	Number of changes generated by ANALYZE

- pg_total_gsi_autovac_tuples(bool)

  Description: The function is not supported in centralized mode.

  Return type: setofrecord

- pg_autovac_status(oid)

  Description: Returns autovac information, such as **nodename**, **nspname**, **relname**, **analyze**, **vacuum**, thresholds for the ANALYZE and VACUUM operations, and the number of analyzed or vacuumed tuples. Only users with the SYSADMIN permission can use this function.

  Return type: setofrecord

  Output parameters: See **Table 7-141**.

Table 7-141 Return parameters

Return Parameter	Type	Description
nspname	text	Name of a namespace.
relname	text	Name of an object, such as a table, an index, or a view.
nodename	text	Node name.
doanalyze	Boolean	Specifies whether to execute **ANALYZE**.
anltuples	bigint	Number of ANALYZE tuples.

Return Parameter	Type	Description
anlthresh	bigint	ANALYZE threshold.
dovacuum	Boolean	Specifies whether to execute **VACUUM**.
vactuples	bigint	Number of VACUUM tuples.
vacthresh	bigint	VACUUM threshold.

- pg_autovac_timeout(oid)

  Description: Returns the number of consecutive timeouts during the autovac operation on a table. If the table information is invalid or the node information is abnormal, **NULL** will be returned.

  Return type: bigint

- pg_stat_get_last_data_changed_time(oid)

  Description: Returns the time when **INSERT**, **UPDATE**, **DELETE**, or **TRUNCATE** was last performed on a table, or the time when **EXCHANGE**, **TRUNCATE**, or **DROP** was last performed on a partition or subpartition. The data in the **last_data_changed** column of the **PG_STAT_ALL_TABLES** view is calculated by using this function. The performance of obtaining the last modification time by using the view is poor when the table has a large amount of data. In this case, you are advised to use functions. The input parameter is a table OID.

  Return type: timestamptz

- pg_stat_set_last_data_changed_time(oid)

  Description: Manually changes the time when **INSERT**, **UPDATE**, **DELETE**, or **EXCHANGE**/**TRUNCATE**/**DROP PARTITION** was last performed.

  Return type: void

- pg_stat_get_last_updated(oid, text)

  Description: Returns the update time of monitoring indicator fields in each table in the DBE_PERF.stat_all_tables, DBE_PERF.stat_all_indexes, DBE_PERF.statio_all_indexes, and DBE_PERF.statio_all_tables views. The input parameter 1 is the table OID, and the input parameter 2 is of the text type. The value can be **"stat_table"**, **"stat_index"**, or **"stat_io"**.

  Return type: timestamptz

- pg_backend_pid()

  Description: Specifies the thread ID of the server thread attached to the current session.

  Return type: integer

- pg_stat_get_activity(integer)

  Description: Returns a record about the backend thread with the specified PID. A record for each active backend in the system is returned if **NULL** is specified. The returned result does not contain the **connection_info** column. The initial user, system administrators and users with the MONADMIN permission can view all data. Common users can only query their own results.

Example:

```
gaussdb=# SELECT * FROM pg_stat_get_activity(139881386280704);
 datid |       pid       | sessionid | usesysid | application_name | state  |
query                      | waiting |          xact_start           |          query_start          |
    backend_start         |         state_change          | client_addr | client_hostname | client_port | enqueue
|     query_id      |   srespool   | global_sessionid | unique_sql_id | trace_id
-------+-----------------+-----------+----------+------------------+--------
+-----------------------------------------------------+---------+-------------------------------
+-------------------------------+-----
-------------------------+-------------------------------+-------------+-----------------+-------------+---------
+------------------+-------------+-----------------+--------------+----------
 16545 | 139881386280704 |        69 |       10 | gsql             | active | select * from
pg_stat_get_activity(139881386280704); | f       | 2022-01-18 19:43:05.167718+08 | 2022-01-18
19:43:05.167718+08 | 2022
-01-18 19:42:33.513507+08 | 2022-01-18 19:43:05.16773+08 |             |               |         -1 |         |
72620543991624410 | default_pool | 1938253334#69#0  |    3751941862 |
(1 row)
```

Return type: setofrecord

Output parameters: See **Table 7-142**.

Table 7-142 Return parameters

Return Parameter	Type	Description
datid	oid	OID of the database that the user session connects to in the backend
pid	bigint	Backend thread ID
sessionid	bigint	Session ID
usesysid	oid	OID of the user logged in to the backend
application_name	text	Name of the application connected to the backend
state	text	Overall status of the backend
query	text	Latest query at the backend. If **state** is **active**, this column shows the ongoing query. In all other states, it shows the last query that was executed.
waiting	Boolean	Specifies whether the backend is currently waiting on a lock. If the backend is currently waiting, the value is **true**.

Return Parameter	Type	Description
xact_start	timestamp with time zone	Time when current transaction was started (null if no transaction is active). If the current query is the first of its transaction, the value of this column is the same as that of the **query_start** column.
query_start	timestamp with time zone	Time when the currently active query was started, or time when the last query was started if **state** is not **active** For a stored procedure, function, or package, the first query time is queried and does not change with the running of statements in the stored procedure.
backend_start	timestamp with time zone	Time when this process was started, that is, when the client connected to the server
state_change	timestamp with time zone	Time when **state** was last modified
client_addr	inet	IP address of the client connected to the backend. If this column is **NULL**, it indicates either the client is connected via a Unix socket on the server or this is an internal thread, such as **AUTOVACUUM**.

Return Parameter	Type	Description
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of **client_addr**. This column will be non-null only for IP connections and only when **log_hostname** is enabled.
client_port	integer	TCP port number that the client uses for communication with this backend (**–1** if a Unix socket is used)
enqueue	text	Unsupported currently
query_id	bigint	ID of a query
srespool	name	Name of the resource pool
global_sessionid	text	Global session ID
unique_sql_id	bigint	Unique SQL statement ID
trace_id	text	Driver-specific trace ID, which is associated with an application request

- pg_stat_get_activity_with_conninfo(integer)

  Description: Returns a record about the backend thread with the specified PID. A record for each active backend in the system is returned if **NULL** is specified. The initial user, system administrators and users with the MONADMIN permission can view all data. Common users can only query their own results.

  Return type: setofrecord

  Return values: See **Table 7-143**.

Table 7-143 Return value description

Return Parameter	Type	Return value description
datid	oid	OID of the database that the user session connects to in the backend
pid	bigint	Backend thread ID
sessionid	bigint	Session ID
usesysid	oid	OID of the user logged in to the backend
application_name	text	Name of the application connected to the backend
state	text	Overall status of the backend
query	text	Latest query at the backend. If **state** is **active**, this column shows the ongoing query. In all other states, it shows the last query that was executed.
waiting	Boolean	Specifies whether the backend is currently waiting on a lock. If yes, the value is **true**.
xact_start	timestamp with time zone	Time when current transaction was started (null if no transaction is active). If the current query is the first of its transaction, the value of this column is the same as that of the **query_start** column.

Return Parameter	Type	Return value description
query_start	timestamp with time zone	Time when the currently active query was started, or time when the last query was started if **state** is not **active** For a stored procedure, function, or package, the first query time is queried and does not change with the running of statements in the stored procedure.
backend_start	timestamp with time zone	Time when this process was started, that is, when the client connected to the server
state_change	timestamp with time zone	Time when **state** was last modified
client_addr	inet	IP address of the client connected to the backend. If this column is **NULL**, it indicates either the client is connected via a Unix socket on the server or this is an internal thread, such as **AUTOVACUUM**.
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of **client_addr**. This column will be non-null only for IP connections and only when **log_hostname** is enabled.
client_port	integer	TCP port number that the client uses for communication with this backend (**–1** if a Unix socket is used)
enqueue	text	Unsupported currently

Return Parameter	Type	Return value description
query_id	bigint	ID of a query
connection_info	text	A string in JSON format recording the driver type, driver version, driver deployment path, and thread owner of the connected database.
srespool	name	Name of the resource pool
global_sessionid	text	Global session ID
unique_sql_id	bigint	Unique SQL statement ID
trace_id	text	Driver-specific trace ID, which is associated with an application request
top_xid	xid	Top-level transaction ID of a transaction.
current_xid	xid	Current transaction ID of a transaction.
xlog_quantity	bigint	Amount of Xlogs currently used by a transaction, in bytes.

- gs_get_explain(integer)

  Description: Returns a running plan for the background thread with the specified PID. The PID cannot be empty. This function takes effect only when the GUC parameter **track_activities** is set to **on**. Only explainable SQL statements whose plans do not contain stream operators are supported. The details are as follows:

  – If the GUC parameter **plan_collect_thresh** is set to **–1**, the return result of the function is always empty.

  – If **plan_collect_thresh** is set to **0**, the current SQL execution time is greater than or equal to the value of **log_min_duration_statement**, and the total number of tuples processed by all operators in the plan is greater than or equal to 10000, the system starts to collect plans in running state. Each time the total number of tuples processed by all operators exceeds 10000, a collection is performed.

  – If **plan_collect_thresh** is set to a value greater than 0, running plans are collected incrementally based on the threshold specified by this parameter.

The return value type is text. The types and meanings of the fields are as follows:

Parameter	Type	Description
Character string of the plans in the running state	text	**A-rows** in the plan string indicates the number of rows returned by the operator in real time.

- pg_stat_get_function_calls(oid)

  Description: Specifies the number of times the function has been called.

  Return type: bigint
- pg_stat_get_function_self_time(oid)

  Description: Specifies the time spent in only this function. The time spent on this function calling other functions is excluded.

  Return type: bigint
- pg_stat_get_backend_idset()

  Description: Sets the number of currently active server threads (from 1 to the number of active server threads).

  Return type: setofinteger
- pg_stat_get_backend_pid(integer)

  Description: Specifies the ID of the given server thread.

  Return type: bigint
- pg_stat_get_backend_dbid(integer)

  Description: Specifies the ID of the database connected to the given server thread.

  Return type: oid
- pg_stat_get_backend_userid(integer)

  Description: Specifies the user ID of the given server thread. This function can be called only by a system administrator.

  Return type: oid
- pg_stat_get_backend_activity(integer)

  Description: Active command of the given server thread, but only if the current user is a system administrator or the same user as that of the session being queried and **track_activities** is enabled.

  Return type: text
- pg_stat_get_backend_waiting(integer)

  Description: Returns true if the given server thread is waiting for a lock, but only if the current user is a system administrator or the same user as that of the session being queried and **track_activities** is enabled.

  Return type: Boolean
- pg_stat_get_backend_activity_start(integer)

  Description: Specifies the time when the given server thread's currently executing query is started only if the current user is the system administrator or the user of the session being queried and **track_activities** is enabled.

Return type: timestamp with time zone

- pg_stat_get_backend_xact_start(integer)

  Description: Specifies the time when the given server thread's currently executing transaction is started only if the current user is the system administrator or the user of the session being queried and **track_activities** is enabled.

  Return type: timestamp with time zone

- pg_stat_get_backend_start(integer)

  Description: Specifies the time when the given server thread is started. If the current user is neither a system administrator nor the same user as that of the session being queried, **NULL** is returned.

  Return type: timestamp with time zone

- pg_stat_get_backend_client_addr(integer)

  Description: Specifies the IP address of the client connected to the given server process. If the connection is over a Unix domain socket, or if the current user is neither a system administrator nor the same user as that of the session being queried, **NULL** will be returned.

  Return type: inet

- pg_stat_get_backend_client_port(integer)

  Description: Specifies the TCP port number of the client connected to the given server process. If the connection is over a Unix domain socket, **-1** will be returned. If the current user is neither a system administrator nor the same user as that of the session being queried, **NULL** will be returned.

  Return type: integer

- pg_stat_get_bgwriter_timed_checkpoints()

  Description: Specifies the time when the background writer thread starts scheduled checkpoints (because **checkpoint_timeout** has expired).

  Return type: bigint

- pg_stat_get_bgwriter_requested_checkpoints()

  Description: Specifies the time when the background writer thread starts checkpoints based on requests from the backend because **checkpoint_segments** has been exceeded or the **CHECKPOINT** command has been executed.

  Return type: bigint

- pg_stat_get_bgwriter_buf_written_checkpoints()

  Description: Specifies the number of buffers written by the background writer thread during checkpoints.

  Return type: bigint

- pg_stat_get_bgwriter_buf_written_clean()

  Description: Specifies the number of buffers written by the background writer thread for routine cleaning of dirty pages.

  Return type: bigint

- pg_stat_get_bgwriter_maxwritten_clean()

  Description: Specifies the time when the background writer thread stops its cleaning scan because it has written more buffers than specified in the **bgwriter_lru_maxpages** parameter.

Return type: bigint

- pg_stat_get_buf_written_backend()

  Description: Specifies the number of buffers written by the backend thread because they need to allocate a new buffer.

  Return type: bigint

- pg_stat_get_buf_alloc()

  Description: Specifies the total number of the allocated buffers.

  Return type: bigint

- pg_stat_clear_snapshot()

  Description: Discards the current statistics snapshot. Only users with the SYSADMIN or MONADMIN permission can execute this function.

  Return type: void

- pg_stat_reset()

  Description: Resets all statistics counters for the current database to zero (requires system administrator permissions).

  Return type: void

- pg_stat_reset_shared(text)

  Description: Resets all statistics counters for the current database in each node in a shared cluster to zero (requires system administrator permissions).

  Return type: void

- pg_stat_reset_single_table_counters(oid)

  Description: Resets statistics for a single table or index in the current database to zero (requires system administrator permissions).

  Return type: void

- pg_stat_reset_single_function_counters(oid)

  Description: Resets statistics for a single function in the current database to zero (requires system administrator permissions).

  Return type: void

- fenced_udf_process(integer)

  Description: Shows the number of local UDF Master and Work threads. If the input parameter is set to **1**, the number of Master threads is queried. If the input parameter is set to **2**, the number of Worker threads is queried. If the input parameter is set to **3**, all Worker threads are terminated.

  Return type: text

- total_cpu()

  Description: Obtains the CPU time used by the current node, in jiffies.

  Return type: bigint

- total_memory()

  Description: Obtains the size of the virtual memory used by the current node, in KB.

  Return type: bigint

- pg_stat_bad_block(text, int, int, int, int, int, timestamp with time zone, timestamp with time zone)

Description: Obtains damage information about pages after the current node is started.

Return type: record

Example:

```
SELECT * FROM pg_stat_bad_block();
```

- pg_stat_bad_block_clear()

  Description: Deletes the page damage information that is read and recorded on the node (requires system administrator permissions).

  Return type: void

- gs_respool_exception_info(pool text)

  Description: Queries the query rule of a specified resource pool.

  Return type: record

- gs_control_group_info(pool text)

  Description: Queries information about Cgroups associated with a resource pool. Only users with the SYSADMIN permission can execute this function.

  Return type: record

  The command output is as follows:

Attribute	Value	Description
name	class_a:workload_a1	Class name and workload name.
class	class_a	Class Cgroup name.
workload	workload_a1	Workload Cgroup name.
type	DEFWD	Cgroup type (**Top**, **CLASS**, **BAKWD**, **DEFWD**, or **TSWD**).
gid	87	Cgroup ID.
shares	30	Percentage of CPU resources to those on the parent node.
limits	0	Percentage of CPU cores to those on the parent node.
rate	0	Allocation ratio in Timeshare.
cpucores	0-3	Number of CPU cores.

- gs_prepared_statements()

  Description: Displays all available prepared statements of all sessions. Only users with the SYSADMIN permission can execute this function. The columns returned by this function are the same as those in **GS_ALL_PREPARED_STATEMENTS**.

  Return type: record

- gs_all_control_group_info()

  Description: Collects information about all Cgroups in the database.

Return type: record

- gs_get_control_group_info()

  Description: Collects information about all Cgroups. For details about the columns returned by the function, see **GS_GET_CONTROL_GROUP_INFO**. Only users with the SYSADMIN permission can execute this function.

  Return type: record

- gs_plan_trace_delete(TIMESTAMPTZ)

  Description: Deletes all plan traces earlier than or equal to **max_time** for the current user. All users can use this function, and as long as no exception occurs during the execution of this function, this function returns **t**.

  Return type: Boolean

- gs_plan_trace_watch_sqlid(bigint)

  Description: Listens to a unique SQL ID for which plan traces are to be generated. The ID is obtained from the **unique_sql_id** column in the dbe_perf.statement system catalog. In addition, this function can be called only by the initial user and users with the SYSADMIN, OPRADMIN, or MONADMIN permission. If no exception occurs during the execution of this function, **t** is returned.

  Return type: Boolean

NOTICE

1. In the database system, the unique SQL ID that is listened to is stored in a cyclic array whose length is 128. If the function is called too frequently, the unique SQL ID that is listened to but does not generate a plan trace may be overwritten.
2. If a unique SQL ID is listened only once, only one plan trace can be generated for the unique SQL ID. If the same unique SQL ID is listened for multiple times, multiple plan traces are generated for the unique SQL ID.

- gs_plan_trace_show_sqlids()

  Description: Queries the list of unique SQL IDs for which plan traces are to be generated in the current system. This function can be called only by the initial user and users with the SYSADMIN, OPRADMIN, or MONADMIN permission.

  Return type: text

NOTICE

1. If there are two unique SQ IDs 730834934 and 730834935 for which plan traces are to be generated in the system, the result of using this function is a character string text "730834934,730834935,".
2. When this function is used, the unique SQL ID that is generating the plan trace cannot be viewed.

- get_instr_workload_info(integer)

  Description: Obtains the transaction volume and time information on the primary database node.

Return type: record

Attribute	Value	Description
resourcepool_oid	10	OID of the resource pool (the logic is equivalent to the load).
commit_counter	4	Number of frontend transactions that were committed.
rollback_counter	1	Number of frontend transactions that were rolled back.
resp_min	949	Minimum response time of frontend transactions (unit: μs).
resp_max	201891	Maximum response time of frontend transactions (unit: μs).
resp_avg	43564	Average response time of frontend transactions (unit: μs).
resp_total	217822	Total response time of frontend transactions (unit: μs).
bg_commit_counter	910	Number of backend transactions that were committed.
bg_rollback_counter	0	Number of backend transactions that were rolled back.
bg_resp_min	97	Minimum response time of backend transactions (unit: μs).
bg_resp_max	678080687	Maximum response time of backend transactions (unit: μs).
bg_resp_avg	327847884	Average response time of backend transactions (unit: μs).
bg_resp_total	298341575300	Total response time of backend transactions (unit: μs).

- pv_instance_time()

  Description: Obtains the time consumed in each execution phase on the current node.

  Return type: record

Stat_name Attribute	Value	Description
DB_TIME	1062385	Total end-to-end wall time consumed by all threads (unit: μs).
CPU_TIME	311777	Total CPU time consumed by all threads (unit: μs).

Stat_name Attribute	Value	Description
EXECUTION_TIME	380037	Total time consumed on the executor (unit: μs).
PARSE_TIME	6033	Total time consumed for parsing SQL statements (unit: μs).
PLAN_TIME	173356	Total time consumed for generating an execution plan (unit: μs).
REWRITE_TIME	2274	Total time consumed on query rewriting (unit: μs).
PL_EXECUTION_TIME	0	Total time consumed for executing PL/SQL statements (unit: μs).
PL_COMPILATION_TIME	557	Total time consumed for compiling SQL statements (unit: μs).
NET_SEND_TIME	1673	Total time consumed for sending data over network (unit: μs).
DATA_IO_TIME	426622	Total time consumed for data read and write (unit: μs).

- DBE_PERF.get_global_instance_time()

  Description: Provides the time consumed in each key phase in the entire database. To query this function, you must have the MONADMIN permission.

  Return type: record

- get_instr_unique_sql()

  Description: Obtains information about execution statements (normalized SQL statements) on the current node as a user with the MONADMIN permission.

  Return type: record

- reset_unique_sql(text, text, bigint)

  Description: Clears the unique SQL statements in the memory of the database node. (The sysadmin or monitor admin permission is required.)

  Return type: Boolean

  Table 7-144 reset_unique_sql parameters

Parameter	Type	Description
scope	text	Clearance scope type. The options are as follows: **'GLOBAL'**: Clears all CNs/DNs. If the value is **'GLOBAL'**, this function can be executed only on the CN. **'LOCAL'**: Clears the current node.

Parameter	Type	Description
clean_type	text	**'BY_USERID'**: Clears unique SQL statements based on user IDs. **'BY_CNID'**: Clears unique SQL statements based on CN IDs. **'ALL'**: Clears all data.
clean_value	int8	Clearance value corresponding to the clearance type. If the second parameter is set to **ALL**, the third parameter does not take effect and can be set to any value.

NOTE

This function involves distributed nodes. Currently, GaussDB is a centralized database, for which the function of the value **global** is the same as that of the value **local** and the second parameter cannot set to be **BY_CNID**.

- get_instr_wait_event(NULL)

  Description: Obtains the statistics on wait events of the current node.

  Return type: record

- get_instr_user_login()

  Description: Obtains the number of user login and logout times on the current node. Only users with the SYSADMIN or MONADMIN permission can execute this function.

  Return type: record

- get_instr_rt_percentile(integer)

  Description: Obtains the response time of 80% and 95% SQL statements in the database.

  Return type: record

- get_node_stat_reset_time()

  Description: Obtains statistics about reset (restart, primary/standby switchover, and database deletion) time of the current node.

  Return type: record

- DBE_PERF.get_global_os_runtime()

  Description: Displays the running status of the current OS. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_os_threads()

  Description: Provides information about the threads under all normal nodes of the entire database. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_summary_workload_sql_count()

  Description: Provides statistics about the number of SELECT, UPDATE, INSERT, DELETE, DDL, DML, and DCL statements of different service loads in the

entire database. To query this function, you must have the MONADMIN permission.

Return type: record

- DBE_PERF.get_summary_workload_sql_elapse_time()

  Description: Provides statistics about the number of SELECT, UPDATE, INSERT, and DELETE statements and response time information (TOTAL, AVG, MIN, and MAX) for different loads in the entire database. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_workload_transaction()

  Description: Obtains the transaction volume and time information on all nodes of the database. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_session_stat()

  Description: Obtains the session status information on all nodes of the database. To query this function, you must have the MONADMIN permission.

  Return type: record

  NOTE

  There are 14 status information items: **commit**, **rollback**, **sql**, **table_scan**, **blocks_fetched**, **physical_read_operation**, **shared_blocks_dirtied**, **local_blocks_dirtied**, **shared_blocks_read**, **local_blocks_read**, **blocks_read_time**, **blocks_write_time**, **sort_imemory**, and **sort_idisk**.

- DBE_PERF.get_global_session_time()

  Description: Provides the time consumed in each key phase of each node in the entire database. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_session_memory()

  Description: Displays statistics about memory usage at the session level on each node in the unit of MB, including all the memory allocated to GaussDB and stream threads on DNs for jobs currently executed by users. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_session_memory_detail()

  Description: Displays statistics about thread memory usage on each node by MemoryContext node. To query this function, you must have the MONADMIN permission.

  Return type: record

- gs_paxos_stat_replication()

  Description: Queries the standby node information on the primary node.

  Return type: SETOF record

  The following table describes return columns.

Column	Type	Description
local_role	text	Role of the sender node.
peer_role	text	Role of the receiver node.
local_dcf_role	text	DCF role of the sender node.
peer_dcf_role	text	DCF role of the receiver node.
peer_state	text	Status of the receiver node.
sender_write_location	text	Location in the Xlog buffer where the sender node is written.
sender_commit_location	text	Barrier reached for the DCF logs of the sender node.
sender_flush_location	text	Location in the Xlog disk where the sender node is written.
sender_replay_location	text	Location where the sender node replays logs.
receiver_write_location	text	Location in the Xlog buffer where the receiver node is written
receiver_commit_location	text	Barrier reached for the DCF logs of the receiver node.
receiver_flush_location	text	Location in the Xlog disk where the receiver node is written.
receiver_replay_location	text	Location where the receiver node replays Xlogs.
sync_percent	text	Synchronization percentage.
dcf_run_mode	int4	DCF synchronization mode.
channel	text	Channel information.

- gs_wlm_get_user_info(int)

  Description: Obtains information about all users. The input parameter is of the int type and can be any int value or **NULL**. Only users with the SYSADMIN permission can execute this function.

  Return type: record

- gs_wlm_readjust_user_space(oid)

  Description: Corrects the storage space usage of all users. Only the administrator can execute this function.

  Return type: record

- gs_wlm_readjust_user_space_through_username(text name)

  Description: Corrects the storage space usage of a specified user. Common users can use this function to modify only their own usage. Only the

administrator can modify the usage of all users. If the value of **name** is **0000**, the usage of all users needs to be modified.

Return type: record

- gs_wlm_readjust_user_space_with_reset_flag(text name, boolean isfirst)

  Description: Corrects the storage space usage of a specified user. If the input parameter **isfirst** is set to **true**, statistics are collected from 0. Otherwise, statistics are collected from the previous result. Common users can use this function to modify only their own usage. Only the administrator can modify the usage of all users. If the value of **name** is **0000**, the usage of all users needs to be modified.

  Return type: record

- gs_wlm_get_session_info()

  Description: This API has been discarded and is unavailable currently.

- gs_wlm_get_user_session_info()

  Description: This API has been discarded and is unavailable currently.

- gs_io_wait_status()

  Description: This API does not support single-node systems or centralized systems and is unavailable currently.

- global_stat_get_hotkeys_info()

  Description: Obtains the statistics on hot keys in the entire database instance. This API does not support single-node systems or centralized systems and is unavailable currently.

- global_stat_clean_hotkeys()

  Description: Clears statistics on hot keys in the entire database instance. This API does not support single-node systems or centralized systems and is unavailable currently.

- DBE_PERF.get_global_session_stat_activity()

  Description: Displays information about threads that are running on each node in the database. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_thread_wait_status()

  Description: Displays the block waiting status of backend threads and auxiliary threads on all nodes. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_operator_history_table()

  Description: Displays the operator-related records (persistent) generated after jobs are executed on the primary database node of the current user. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_operator_history()

  Description: Displays the operator-related records generated after jobs are executed on the primary database node of the current user. To query this function, you must have the SYSADMIN or MONADMIN permission.

Return type: record

- DBE_PERF.get_global_operator_runtime()

  Description: Displays real-time operator-related records of jobs executed on the primary database node of the current user. To query this function, you must have the SYSADMIN or MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_statement_complex_history()

  Description: Displays the historical records of complex queries on the primary database node of the current user. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_statement_complex_history_table()

  Description: Displays the historical records (persistent) of complex queries on the primary database node of the current user. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_statement_complex_runtime()

  Description: Displays the real-time information of complex queries on the primary database node of the current user. To query this function, you must have the SYSADMIN or MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_memory_node_detail()

  Description: Displays the memory usage of a certain database on all nodes. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_shared_memory_detail()

  Description: Displays the usage information about all the shared memory contexts of all nodes. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_statio_all_indexes()

  Description: Displays statistics about each index displayed in a row in the current database, showing I/O statistics about accesses to that specific index. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_summary_stat_all_tables()

  Description: Displays statistics about a row in each table (including the TOAST table) in the aggregated data of each node. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_stat_all_tables()

  Description: Displays statistics about a row in each table (including the TOAST table) of data on each node. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_local_toastname_and_toastindexname()

  Description: Provides the mapping between the name and index of the local TOAST table and its associated table. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_summary_statio_all_indexes()

  Description: Collects statistics about each index displayed in a row in the current databases of all nodes and displays the I/O statistics of a specific index. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_statio_all_sequences()

  Description: Provides I/O status information about all sequences in the namespace. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_statio_all_tables()

  Description: Displays the I/O statistics about each table in databases on each node. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_summary_statio_all_tables()

  Description: Collects I/O statistics about each table in the database. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_local_toast_relation()

  Description: Provides the mapping between the name of the local TOAST table and its associated table. After a cluster is created, you must have the MONADMIN permission by default to query this function.

  Return type: record

- DBE_PERF.get_global_statio_sys_indexes()

  Description: Displays the I/O status information about all system catalog indexes in namespaces on each node. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_summary_statio_sys_indexes()

  Description: Collects the I/O status information about all system catalog indexes in namespaces on each node. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_statio_sys_sequences()

  Description: Provides the I/O status information about all the system sequences in the namespace. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_statio_sys_tables()

Description: Provides I/O status information about all system catalogs in namespaces on each node. To query this function, you must have the MONADMIN permission.

Return type: record

- DBE_PERF.get_summary_statio_sys_tables()

  Description: Displays the I/O status information of all system catalogs in the namespace in the database. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_statio_user_indexes()

  Description: Displays the I/O status information about all user relationship table indexes in namespaces on each node. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_summary_statio_user_indexes()

  Description: Displays the I/O status information about all user relationship table indexes in namespaces in the database. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_statio_user_sequences()

  Description: Displays the I/O status information about all user sequences in the namespace of each node. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_statio_user_tables()

  Description: Displays the I/O status information about all user relationship tables in namespaces on each node. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_summary_statio_user_tables()

  Description: Displays the I/O status information about all user relationship tables in namespaces in the database. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_stat_all_indexes()

  Description: Displays statistics of each index in databases on all nodes. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_summary_stat_all_indexes()

  Description: Collects statistics of each index in all databases on all nodes. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_stat_sys_tables()

Description: Displays statistics about the system catalogs of all the namespaces in **pg_catalog** and **information_schema** schemas on each node. To query this function, you must have the MONADMIN permission.

Return type: record

- DBE_PERF.get_summary_stat_sys_tables()

  Description: Collects statistics about the system catalogs of all the namespaces in pg_catalog and information_schema schemas on each node. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_stat_sys_indexes()

  Description: Displays index status information about all the system catalogs in the pg_catalog and information_schema schemas on each node. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_summary_stat_sys_indexes()

  Description: Collects statistics about index status information about all the system catalogs in the pg_catalog and information_schema schemas on each node. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_stat_user_tables()

  Description: Displays the status information about customized ordinary tables in all namespaces. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_summary_stat_user_tables()

  Description: Collects statistics about the status information about customized ordinary tables in all namespaces. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_stat_user_indexes()

  Description: Displays the status information about the index of customized ordinary tables in all databases. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_summary_stat_user_indexes()

  Description: Collects statistics about the status information about the index of customized ordinary tables in all databases. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_stat_database()

  Description: Displays database statistics of all nodes. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_stat_database_conflicts()

Description: Collects statistics on the database of all nodes. To query this function, you must have the MONADMIN permission.

Return type: record

- DBE_PERF.get_global_stat_xact_all_tables()

  Description: Displays transaction status information about all ordinary tables and TOAST tables in all namespaces. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_summary_stat_xact_all_tables()

  Description: Collects statistics about transaction status information about all ordinary tables and TOAST tables in all namespaces. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_stat_xact_sys_tables()

  Description: Displays transaction status information about all system catalogs in namespaces on all nodes. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_summary_stat_xact_sys_tables()

  Description: Collects statistics about transaction status information about all system catalogs in namespaces on all nodes. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_stat_xact_user_tables()

  Description: Displays the transaction status information of the user tables in the namespaces on all nodes. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_summary_stat_xact_user_tables()

  Description: Collects statistics about the transaction status information of the user tables in the namespaces on all nodes. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_stat_user_functions()

  Description: Displays the transaction status information of customized functions in the namespaces on all nodes. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_stat_xact_user_functions()

  Description: Collects statistics about the transaction status information of customized functions in the namespaces on all nodes. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_stat_bad_block()

Description: Displays information about table and index read failures on all nodes. To query this function, you must have the MONADMIN permission.

Return type: record

- DBE_PERF.get_global_file_redo_iostat()

  Description: Collects statistics on information about table and index read failures on all nodes. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_file_iostat()

  Description: Displays statistics about data file I/Os on all nodes. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_locks()

  Description: Displays lock information of all nodes. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_replication_slots()

  Description: Displays logical replication information on all nodes. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.GET_GLOBAL_PARALLEL_DECODE_STATUS()

  Description: Displays parallel decoding information of replication slots on the current node. To query this function, you must have the MONADMIN permission. The return value is the same as that of the view **GLOBAL_PARALLEL_DECODE_STATUS**.

  Return type: record

- DBE_PERF.GET_GLOBAL_PARALLEL_DECODE_THREAD_INFO()

  Description: Displays parallel decoding thread information of replication slots on the current node. To query this function, you must have the MONADMIN permission. The return value is the same as that of the view **GLOBAL_PARALLEL_DECODE_THREAD_INFO**.

  Return type: record

- DBE_PERF.get_global_bgwriter_stat()

  Description: Displays statistics about the background writer thread's activities on all nodes. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_replication_stat()

  Description: Displays information about log synchronization status on each node, such as the locations where the sender sends logs and where the receiver receives logs. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_transactions_running_xacts()

Description: Displays information about running transactions on each node. To query this function, you must have the MONADMIN permission.

Return type: record

- DBE_PERF.get_summary_transactions_running_xacts()

  Description: Collects statistics of information about running transactions on each node. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_transactions_prepared_xacts()

  Description: Displays information about transactions that are currently prepared for two-phase commit on each node. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_summary_transactions_prepared_xacts()

  Description: Collects statistics information about transactions that are currently prepared for two-phase commit on each node. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_summary_statement()

  Description: Displays the status information of the historically executed statements on each node. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_statement_count()

  Description: Displays the number of SELECT, UPDATE, INSERT, and DELETE statements and response time information (**TOTAL**, **AVG**, **MIN**, and **MAX**) on each node. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_config_settings()

  Description: Displays the wait event status information on each node. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_global_wait_events()

  Description: Displays the wait event status information on each node. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_statement_responsetime_percentile()

  Description: Obtains the response time distribution for 80% and 95% SQL statements of the database. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.get_summary_user_login()

Description: Collects statistics about number of user login and logout times on each node in the database. To query this function, you must have the MONADMIN permission.

Return type: record

- DBE_PERF.get_global_record_reset_time()

  Description: Displays the statistics about reset (restart, primary/standby switchover, and database deletion) time of the database. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.standby_statement_history(only_slow[, time1, time2])

  Description: Queries full SQL statements on the standby node. The primary node queries full SQL statements using the statement_history table, while the standby node queries using this function. To query this function, you must have the MONADMIN permission.

  Parameters: See **Table 7-145**.

  Return type: record, which is the same as that in the statement_history table.

Table 7-145 standby_statement_history parameters

Parameter	Type	Description
only_slow	Boolean	Specifies whether to query only slow SQL statements. **true**: yes, which is equivalent to **select .. where is_slow_sql = true;**. **false** or **NULL** indicates that all SQL statements are queried, that is, **is_slow_sql** is not used as a filtering condition.
time1	timestamptz	Minimum time specified by **finish_time** for querying SQL statements. This parameter is optional.
time2	timestamptz	Maximum time specified by **finish_time** for querying SQL statements. This parameter is optional.

NOTE

- The two time parameters **time1** and **time2** indicate the time segment to which **finish_time** of the queried SQL statement belongs. They indicate the start time and end time respectively. If **NULL** or no value is entered, there is no limit. The function of **time1** and **time2** is the same as that of **select .. where finish_time between time1 and time2**.
- The data generated from this function on the standby node is not stored in a table, and there is no index on the **start_time** column. You are advised to use the parameter to search for **finish_time**.
- Full/Slow SQL statements on the standby node are written to disks asynchronously. Therefore, the storage of user SQL information may be delayed. You are advised to query this API to expand the query time range.

- DBE_PERF.track_memory_context(context_list text)

  Description: Sets the memory context whose memory application details need to be collected. The input parameter is the memory context names, which are separated by commas (,), for example, **ThreadTopMemoryContext, SessionCacheMemoryContext**. Note that the memory context names are context-sensitive. In addition, the length of a single memory context is 63, and the excess part is truncated. The maximum number of memory contexts that can be collected at a time is 16. If the number of memory contexts exceeds 16, the setting fails. Each time this function is called, the previous statistics result is cleared. When the input parameter is set to **""**, the statistics function is disabled. To query this function, you must have the MONADMIN permission.

  Return type: Boolean

- DBE_PERF.track_memory_context_detail()

  Description: Obtains the memory application details of the memory context specified by the DBE_PERF.track_memory_context function. For details, see the DBE_PERF.track_memory_context_detail view. To query this function, you must have the MONADMIN permission.

  Return type: record

- pg_stat_get_mem_mbytes_reserved(tid)

  Description: Collects statistics on variables related to resource management, which is used only for fault locating.

  Parameter: thread ID

  Return type: text

- pg_stat_get_file_stat()

  Description: Records statistics about data file I/Os to indicate I/O performance and detect performance problems such as abnormal I/O operations.

  Return type: record

- pg_stat_get_redo_stat()

  Description: Displays statistics on the replay of session thread logs.

  Return type: record

- pg_stat_get_status(int8)

  Description: Tests the block waiting status about the backend thread and auxiliary thread of the current instance.

  Return type: record

- get_local_rel_iostat()

  Description: Queries the accumulated I/O status of data files on the current node.

  Return type: record

- DBE_PERF.get_global_rel_iostat()

  Description: Displays statistics about data file I/Os on all nodes. To query this function, you must have the MONADMIN permission.

  Return type: record

- DBE_PERF.global_threadpool_status()

Description: Displays the status of worker threads and sessions in thread pools on all nodes. The **GLOBAL_THREADPOOL_STATUS** column is returned by the function. To query the function, you must have the MONADMIN permission.

Return type: record

- pv_os_run_info()

  Description: Displays the running status of the current OS. For details about the columns, see **GS_OS_RUN_INFO**.

  Parameter: nan

  Return type: SETOF record

- pv_session_stat()

  Description: Collects session status information by session thread or AutoVacuum thread. For details about the columns, see **GS_SESSION_STAT**.

  Parameter: **nan**

  Return type: SETOF record

- pv_session_time()

  Description: Collects statistics on the running time of session threads and the time consumed in each execution phase. For details about the columns, see **GS_SESSION_TIME**.

  Parameter: **nan**

  Return type: SETOF record

- pg_stat_get_db_temp_bytes()

  Description: Collects statistics on the total amount of data written to temporary files through database query. All temporary files are counted, regardless of why the temporary file was created, and regardless of the **log_temp_files** setting.

  Parameter: **oid**

  Return type: bigint

- pg_stat_get_db_temp_files()

  Description: Queries the number of temporary files created in the database. All temporary files are counted, regardless of why the temporary file was created (for example, sorting or hashing), and regardless of the **log_temp_files** setting.

  Parameter: **oid**

  Return type: bigint

- remote_candidate_stat()

  Description: Displays the checkpoint information and log flushing information about all instances in the database (except the current node). Centralized systems are not supported.

  Return type: record

- remote_ckpt_stat()

  Description: Displays the checkpoint information and log flushing information about all instances in the database (except the current node). Centralized systems are not supported.

  Return type: record

- remote_single_flush_dw_stat()

  Description: Displays the single-page doublewrite file status of all instances in the database (except the current node). Centralized systems are not supported.

  Return type: record

- remote_double_write_stat()

  Description: Displays doublewrite file status of all instances in the database (except the current node). Centralized systems are not supported.

  Return type: record

- remote_pagewriter_stat()

  Description: Displays the page flushing information and checkpoint information about all instances in the database (except the current node). Centralized systems are not supported.

  Return type: record

- remote_recovery_status()

  Description: Displays log flow control information about the primary and standby nodes (except the current node). Centralized systems are not supported.

  Return type: record

- remote_redo_stat()

  Description: Displays the log replay status of all instances in the database (except the current node). Centralized systems are not supported.

  Return type: record

- DBE_PERF.gs_stat_activity_timeout(int)

  Description: Obtains information about query jobs whose execution time exceeds the timeout threshold on the current node. The correct result can be returned only when the GUC parameter **track_activities** is set to **on**. The timeout threshold ranges from 0 to 2147483. To query this function, you must have the MONADMIN permission.

  Return type: SETOF record

Name	Type	Description
database	name	Name of the database to which a user session is connected.
pid	bigint	Backend thread ID.
sessionid	bigint	Session ID
usesysid	oid	OID of the user logged in to the backend.
application_name	text	Name of the application connected to the backend.
query	text	Query that is being executed on the backend.
xact_start	timestamptz	Time when the current transaction is started.

Name	Type	Description
query_start	timestamptz	Time when the current query starts For a stored procedure, function, or package, the first query time is queried and does not change with the running of statements in the stored procedure.
query_id	bigint	Query statement ID.

- gs_wlm_user_resource_info(name text)

  Description: Queries a user's resource quota and resource usage. Common users can query only their own information. Administrators can query information about all users.

  Return type: record

- local_redo_time_count()

  Description: Returns the time consumption statistics on each process of each replayer thread on the current node (valid data exists only on the standby node).

  The return values are as follows:

Column	Description
thread_name	Thread name.
step1_total	Total duration of step 1. The process of each thread is as follows: • Ultimate RTO: **redo batch**: obtains a log from a queue. **redo manager**: obtains a log from a queue. **redo worker**: obtains a log from a queue. **trxn manager**: reads a log from a queue. **trxn worker**: reads a log from a queue. **read worker**: reads an Xlog page (overall) from a file. **read page worker**: obtains a log from a queue. **startup**: obtains a log from a queue. • Parallel replay: **page redo**: obtains a log from a queue. **startup**: reads a log.

Column	Description
step1_count	Number of accumulated execution times of step 1.
step2_total	Total duration of step 2. The process of each thread is as follows: • Ultimate RTO: **redo batch**: processes logs (overall). **redo manager**: processes logs (overall). **redo worker**: processes logs (overall). **trxn manager**: processes logs (overall). **trxn worker**: processes logs (overall). **read worker**: specifies the time required for reading the Xlog page. **read page worker**: generates and sends LSN forwarders. **startup**: checks whether to replay to the specified position. • Parallel replay: **page redo**: processes logs (overall). **startup**: checks whether to replay to the specified position.
step2_count	Number of accumulated execution times of step 2.

Column	Description
step3_total	Total duration of step 3. The process of each thread is as follows: ● Ultimate RTO: **redo batch**: updates the standby state. **redo manager**: processes data logs. **redo worker**: replays page logs (overall). **trxn manager**: updates the flushing LSN. **trxn worker**: replays logs. **read worker**: pushes the Xlog segment. **read page worker**: obtains a new item. **startup**: collects statistics on the wait time of delayed replay feature. ● Parallel replay: **page redo**: updates the standby state. **startup**: collects statistics on the wait time of delayed replay feature.
step3_count	Number of accumulated execution times of step 3.

Column	Description
step4_total	Total duration of step 4. The process of each thread is as follows: ● Ultimate RTO: **redo batch**: parses Xlogs. **redo manager**: processes DDL operations. **redo worker**: reads data pages. **trxn manager**: synchronizes the wait time. **trxn worker**: updates the LSN of the current thread. **read page worker**: stores logs in the distribution thread. **startup**: distributes logs (overall). ● Parallel replay: **page redo**: replays undo logs. **startup**: distributes logs (overall).
step4_count	Number of accumulated execution times of step 4.
step5_total	Total duration of step 5. The process of each thread is as follows: ● Ultimate RTO: **redo batch**: distributes to the redo manager. **redo manager**: distributes logs to redo workers. **redo worker**: replays data page logs. **trxn manager**: distributes data to the trxn worker. **trxn worker**: forcibly synchronizes the wait time. **read page worker**: updates the LSN of the current thread. **startup**: decodes logs. ● Parallel replay: **page redo**: replays shared transaction logs. **startup**: replays logs.
step5_count	Number of accumulated execution times of step 5.

Column	Description
step6_total	Total duration of step 6. The process of each thread is as follows: ● Ultimate RTO: **redo worker**: replays non-data page logs. **trxn manager**: updates global LSNs. **redo manager**: performs CRC verification for data page logs stored in hash tables. ● Parallel replay: **page redo**: replays synctrxn logs. **startup**: forcibly synchronizes the wait time.
step6_count	Number of accumulated execution times of step 6.
step7_total	Total duration of step 7. The process of each thread is as follows: ● Ultimate RTO: **redo manager**: creates tablespaces. **redo worker**: updates FSM. ● Parallel replay: **page redo**: replays a single log.
step7_count	Number of accumulated execution times of step 7.
step8_total	Total duration of step 8. The process of each thread is as follows: ● Ultimate RTO: **redo worker**: forcibly synchronizes the wait time. ● Parallel replay: **page redo**: replays all workers do logs.
step8_count	Number of accumulated execution times of step 8.

Column	Description
step9_total	Total duration of step 9. The process of each thread is as follows: ● Ultimate RTO: **redo manager**: distributes logs to the page redo thread. ● Parallel replay: **page redo**: replays multi-worker do logs.
step9_count	Number of accumulated execution times of step 9.

- local_xlog_redo_statics()

  Description: Returns the statistics on each type of logs that have been replayed on the current node (valid data exists only on the standby node).

  The return values are as follows:

  Table 7-146 local_xlog_redo_statics parameters

Column	Description
xlog_type	Log types.
rmid	Resource manager ID.
info	Xlog operation.
num	Number of logs.
extra	Valid values are available for page replay logs and xact logs. ● Number of pages read from the disk if the log is of the page type. ● Number of deleted files if the log is of the xact type.

- gs_get_shared_memctx_detail(text)

  Description: Returns the memory application details of the specified memory context, including the file, line number, and size of each memory application (the size of the same line in the same file is accumulated). Only the memory context queried through the pg_shared_memory_detail view can be queried. The input parameter is the memory context name (that is, the **contextname** column in the result returned by the pg_shared_memory_detail view). To query this function, you must have the SYSADMIN or MONADMIN permission.

  Return type: SETOF record

Name	Type	Description
file	text	Name of the file where the memory is applied for.
line	int8	Line number of the code in the file where the requested memory is located.
size	int8	Size of the applied memory. The value is accumulated if the memory is applied for multiple times in the same line of the same file.

NOTE

This view is not supported in the Lite release version.

- gs_get_session_memctx_detail(text)

  Description: Returns the memory application details of the specified memory context, including the file, line number, and size of each memory application (the size of the same line in the same file is accumulated). This parameter is valid only in thread pool mode. Only the memory context queried through the gs_session_memory_context view can be queried. The input parameter is the memory context name (that is, the **contextname** column in the result returned by the gs_session_memory_context view). To query this function, you must have the SYSADMIN or MONADMIN permission.

  Return type: SETOF record

Name	Type	Description
file	text	Name of the file where the memory is applied for.
line	int8	Line number of the code in the file where the requested memory is located.
size	int8	Size of the allocated memory, in bytes. The value is accumulated if the memory is allocated for multiple times to the same line of the same file.

NOTE

This view takes effect only in thread pool mode and is not supported in the Lite release version.

- gs_get_history_memory_detail(cstring)

  Description: Queries historical memory snapshot information. The input parameter type is cstring. The value can be **NULL** or the name of the memory snapshot log file.

  a. If the value of the input parameter is **NULL**, the list of all memory snapshot log files on the current node is displayed.

b. If the value of the input parameter is the name of the memory snapshot log file in the list queried in **a**, the detailed information about the memory snapshot recorded in the log file is displayed.

c. If you enter any other input parameter, the system displays a message indicating that the input parameter is incorrect or the file fails to be opened.

To query this function, you must have the SYSADMIN or MONADMIN permission.

Return type: text

Name	Type	Description
memory_info	text	Memory information. If the input parameter of the function is set to **NULL**, the memory snapshot file list is displayed. If the input parameter is set to the name of the memory snapshot file, the content of the file is displayed.

- gs_stack()

  Description: Displays the call stack of a thread. To query this function, you must have the SYSADMIN or MONADMIN permission.

  Parameter: tid, which indicates the thread ID. **tid** is an optional parameter. If it is specified, the function returns the call stack of the thread corresponding to **tid**. If it is not specified, the function returns the call stacks of all threads.

  Return value: If **tid** is specified, the return value is of the TEXT type. If **tid** is not specified, the return value is a SETOF record.

  Example:

  Obtain the call stack of a specified thread.

```
gaussdb=# SELECT * FROM gs_stack(139663481165568);
                          gs_stack
------------------------------------------------------------------
 __poll + 0x2d                                                   +
 WaitLatchOrSocket(Latch volatile*, int, int, long) + 0x29f      +
 WaitLatch(Latch volatile*, int, long) + 0x2e                    +
 JobScheduleMain() + 0x90f                                       +
 int GaussDbThreadMain<(knl_thread_role)9>(knl_thread_arg*) + 0x456+
 InternalThreadFunc(void*) + 0x2d                                +
 ThreadStarterFunc(void*) + 0xa4                                 +
 start_thread + 0xc5                                           +
 clone + 0x6d                                                  +
(1 row)
```

  Obtain the call stacks of all threads.

```
gaussdb=# select * from gs_stack();
-[ RECORD
1 ]-----------------------------------------------------------------------------------------------------------
tid   | 139670364324352
lwtid | 308
stack | __poll + 0x2d
      | CommWaitPollParam::caller(int (*)(pollfd*, unsigned long, int), unsigned long) + 0x34
      | int comm_socket_call<CommWaitPollParam, int (*)(pollfd*, unsigned long,
int)>(CommWaitPollParam*, int (*)(pollfd*, unsigned long
, int)) + 0x28
      | comm_poll(pollfd*, unsigned long, int) + 0xb1
      | ServerLoop() + 0x72b
      | PostmasterMain(int, char**) + 0x314e
      | main + 0x617
      | __libc_start_main + 0xf5
```



```
     | 0x55d38f8db3a7
[ RECORD 2 ]----------------------------------------------------------------------------------------------------------
tid   | 139664851859200
lwtid | 520
stack | __poll + 0x2d
     | WaitLatchOrSocket(Latch volatile*, int, int, long) + 0x29f
     | SysLoggerMain(int) + 0xc86
     | int GaussDbThreadMain<(knl_thread_role)17>(knl_thread_arg*) + 0x45d
     | InternalThreadFunc(void*) + 0x2d
     | ThreadStarterFunc(void*) + 0xa4
     | start_thread + 0xc5
     | clone + 0x6d
```

- gs_perf_start()

  Description: Calls perf_event_open to collect the call stack of each thread and the running time of each function. To query this function, you must have the SYSADMIN or MONADMIN permission.

  The parameters are described as follows.

Parameter	Description	Type	Value Range
duration	Stack collection duration, in seconds. If a floating-point number is entered, the first digit after the decimal point is rounded off.	integer	1~60
freq	Stack collection frequency. This parameter is optional. The unit is Hz. The default value is **100**.	integer	10~1000

  Return type: text

  Example:

  Set the collection frequency to 100 Hz and collect stack information for 10s.

```
gaussdb=# SELECT * FROM gs_perf_start(10, 100);
   gs_perf_start
---------------------
 Perf start succeed.
(1 row)
```

CAUTION

During data collection, gs_perf_start needs to apply for a ring buffer. The buffer size is controlled by **/proc/sys/kernel/perf_event_mlock_kb** in the OS. If the error message "perf mmap failed" is displayed during the collection, adjust the size of **/proc/sys/kernel/perf_event_mlock_kb** and start the collection again.

- gs_perf_query()

  Description: Collects the function call stacks of each thread, sums up the function running time, and displays the collection result. To query this function, you must have the SYSADMIN or MONADMIN permission.

  Parameter: nan

  Return type: SETOF record

The following table describes the fields returned by the function.

Name	Type	Description
backtrace	text	Stack name (in a tree structure).
period	bigint	Execution time of a stack.
level	integer	Level of the stack call tree where a stack is located.
sequence	integer	Sequence in the stack call tree after sorting.
thread_name	text	Name of the thread where a stack is located.
overhead	float	Percentage of the stack execution time.

Example:

Query the collected stack information.

```
gaussdb=# SELECT * FROM gs_perf_query() WHERE overhead > 2 AND level < 10;
                                backtrace                                |   period    | level | sequence |
thread_name  | overhead
-------------------------------------------------------------------------------------+-------------+-------+----------
+---------------+----------
 root                                                                    +| 74140000000 |     0 |        1 | root          |
100
                                                                        |             |       |          |               |
    worker                                                                +| 69930000000 |     1 |        2 | worker
|    94.32
                                                                        |             |       |          |               |
    |── start_thread                                                      +| 67620000000 |     2 |        3 | worker
|    91.21
                                                                        |             |       |          |               |
    |   └── ThreadStarterFunc                                             +| 67620000000 |     3 |        4 |
worker       |    91.21
                                                                        |             |       |          |               |
    |       └── internal_thread_func                                      +| 67620000000 |     4 |        5 |
worker       |    91.21
                                                                        |             |       |          |               |
    |         └── int gauss_db_thread_main                                +| 67620000000 |     5 |        6 |
worker       |    91.21
                                                                        |             |       |          |               |
     |            └── backend_run                                         +| 67620000000 |     6 |        7 |
worker       |    91.21
                                                                        |             |       |          |               |
    |              |── PostgresMain                                       +| 67520000000 |     7 |        8 | worker
|    91.07
                                                                        |             |       |          |               |
    |              |  |── exec_simple_query                               +| 64800000000 |     8 |        9 |
worker       |     87.4
                                                                        |             |       |          |               |
    |              |  |  |── OpFusion::opfusion_process                    +| 30130000000 |     9 |       10 |
worker       |    40.64
                                                                        |             |       |          |               |
    |              |  |  |── pg_analyze_and_rewrite                      +| 11290000000 |     9 |     1405 |
worker       |    15.23
                                                                        |             |       |          |               |
    |              |  |  |── pg_plan_queries                             +|  9550000000 |     9 |     2660 |
worker       |    12.88
                                                                        |             |       |          |               |
    |              |  |  |── PortalRun                                   +|  4680000000 |     9 |     4310 | worker
```



```
|    6.31
                                                                |          |      |        |           |
    |              |  |  |── finish_xact_command                +|  3120000000 |     9 |     4923 |
worker       |     4.21
                                                                |          |      |        |           |
    |              |  |  |── pg_parse_query                     +|  1520000000 |     9 |     5262 |
worker       |     2.05
                                                                |          |      |        |           |
    |              |  |  |── OpFusion::opfusion_factory         +|  1500000000 |     9 |     5374 |
worker       |     2.02
                                                                |          |      |        |           |
    txnsnapworker                                               +|  1640000000 |     1 |     6770 |
txnsnapworker |     2.21
                                                                |          |      |        |           |
    |── start_thread                                            +|  1620000000 |     2 |     6771 |
txnsnapworker |     2.19
                                                                |          |      |        |           |
    |  └── ThreadStarterFunc                                    +|  1620000000 |     3 |     6772 |
txnsnapworker |     2.19
                                                                |          |      |        |           |
    |     └── internal_thread_func                              +|  1620000000 |     4 |     6773 |
txnsnapworker |     2.19
                                                                |          |      |        |           |
    |        └── int gauss_db_thread_main                       +|  1620000000 |     5 |     6774 |
txnsnapworker |     2.19
                                                                |          |      |        |           |
    |           └── txn_snap_cap_worker_main                    +|  1620000000 |     6 |     6775 |
txnsnapworker |     2.19
                                                                |          |      |        |           |
    |             └── PostgresInitializer::InitTxnSnapWorker    +|  1620000000 |     7 |     6776 |
txnsnapworker |     2.19
                                                                |          |      |        |           |
    |              |── PostgresInitializer::SetDatabase         +|  1600000000 |     8 |     6777 |
txnsnapworker |     2.16
                                                                |          |      |        |           |
    |              |  └── PostgresInitializer::SetDatabaseByName+|  1600000000 |     9 |     6778 |
txnsnapworker |     2.16
```

- gs_perf_report()

  Description: Generates a graphical flame graph file based on the stack data collected by executing the gs_perf_start function and saves the file in the *$GAUSSLOG*/**gs_flamegraph/***{datanode}* directory. To query this function, you must have the SYSADMIN or MONADMIN permission.

  Parameter: nan

  Return type: text

  Example:

  Generate a flame graph file.

```
gaussdb=# SELECT * FROM gs_perf_report();
                        gs_perf_report
-----------------------------------------------------------------------
 Perf report succeed, flamegraph file: flamegraph-2023-11-26_164802.html
(1 row)
```

- gs_perf_clean()

  Description: Clears data generated by **perf**. To query this function, you must have the SYSADMIN or MONADMIN permission.

  Parameter: nan

  Return type: text

  Example:

  Clear data generated by perf.

```
gaussdb=# SELECT * FROM gs_perf_clean();
   gs_perf_clean
```



```
---------------------
 Perf clean succeed.
(1 row)
```

- gs_get_thread_memctx_detail(tid,text)

  Description: Returns the memory application details of the specified memory context, including the file, line number, and size of each memory application (the size of the same line in the same file is accumulated). Only the memory context queried through the gs_thread_memory_context view is supported. The first input parameter is the thread ID (the **tid** column of the data returned by the gs_thread_memory_context), and the second parameter is the memory context name (the **contextname** column of the data returned by gs_thread_memory_context). To query this function, you must have the SYSADMIN or MONADMIN permission.

  Return type: SETOF record

Name	Type	Description
file	text	Name of the file where the memory is applied for.
line	int8	Line number of the code in the file where the requested memory is located.
size	int8	Size of the allocated memory, in bytes. The value is accumulated if the memory is allocated for multiple times to the same line of the same file.

  NOTE

  This view is not supported in the Lite release version.

- gs_tpworker_execstmt_stat()

  Description: Displays the runtime information of a statement. If the SYSADMIN or MONADMIN user runs the statement, the information about all the statements that are being executed is displayed. Common users can query only the information about the SQL statements executed by themselves.

  Return type: SETOF record

Name	Type	Description
db_oid	oid	OID of the database that the user session connects to in the backend.
db_name	name	Name of the database that the user session connects to in the backend.
threadpool_worker	varchar	NUMA group to which a thread belongs and thread ID. The format is *numagroup_threadid*.
thread_id	bigint	Thread ID.
session_id	bigint	Session ID.

Name	Type	Description
query_id	bigint	ID of the SQL statement that is being executed.
query_text	text	Content of the SQL statement that is being executed
unique_sql_id	bigint	Unique ID generated by the SQL statement
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of **client_addr**. This column will be non-null only for IP connections and only when **log_hostname** is enabled.
client_app_name	text	Name of the client app.
stmt_slow_time_threshold	int	Preset timeout interval for marking an SQL statement as a slow SQL statement, in milliseconds.
stmt_start_time	timestamp with time zone	Time when the statement starts to be executed For a stored procedure, function, or package, the first query time is queried and does not change with the running of statements in the stored procedure.
stmt_elapse_time	int	Time elapsed since the query starts.
stmt_control_status	varchar	Current statement state. • **Waiting**: The session access is successful but it is not executed by the thread. • **Running**: The current statement is executed properly. • **Control**: The current statement enters the resource control phase.
stmt_control_rule	text	Slow SQL control rule corresponding to the current language.
stmt_control_iostat	text	IOPS value and maximum IOPS of the current statement. The format is *curVal*/*maxVal*.
stmt_control_memstat	text	This field is reserved and is not supported currently.
stmt_control_cpustat	text	This field is reserved and is not supported currently.

Name	Type	Description
stmt_control_netstat	text	This field is reserved and is not supported currently.

- gs_tpworker_execslot_stat()

  Description: Displays the thread running information. If the SYSADMIN or MONADMIN user runs the command, information about all threads is displayed. Common users can query only information about the threads where the SQL statements executed by themselves are located.

  Return type: SETOF record

Name	Type	Description
numagroup	int	NUMA group to which the current thread belongs.
worker_id	int	Thread ID of the current thread.
worker_bind_type	text	Thread binding mode. The value can be **numabind**, **cpubind**, **allbind**, or **nobind**.
worker_cpu_affinity	text	Affinity between threads and CPU cores, that is, the range of CPU cores that can be scheduled by threads.
worker_status	varchar	Current thread status: • **Waiting**: The session access is successful but it is not executed by the thread. • **Running**: The current statement is executed properly. • **Control**: The current statement enters the resource control phase.
served_query_id	bigint	ID of the SQL statement that is being executed.
served_query_text	text	Content of the SQL statement that is being executed.

- gs_session_all_settings(sessionid bigint)

  Description: Queries the full GUC parameter settings of the session corresponding to the session ID on the local node. To execute this function, you must have the SYSADMIN or MONADMIN permission.

  Input parameter description: **sessionid** indicates the session ID.

  Return type: SETOF record

  The following table describes return fields.

Name	Type	Description
name	text	Parameter name.

Name	Type	Description
setting	text	Current parameter value.
unit	text	Implicit unit of a parameter.

Example:

```
gaussdb=# SELECT sessionid FROM pg_stat_activity WHERE usename = 'testuser';
 sessionid
-----------
    788861
(1 row)

gaussdb=# SELECT * FROM gs_session_all_settings(788861) WHERE name = 'work_mem';
   name   | setting | unit
----------+---------+------
 work_mem | 131072  | kB
(1 row)
```

- gs_session_all_settings()

  Description: Queries full GUC parameter settings of all sessions on the local node. To execute this function, you must have the SYSADMIN or MONADMIN permission.

  Return type: SETOF record

Name	Type	Description
sessionid	bigint	Session ID.
pid	bigint	Backend thread ID.
name	text	Parameter name.
setting	text	Current parameter value.
unit	text	Implicit unit of a parameter.

  Example:

```
gaussdb=# SELECT * FROM gs_session_all_settings() WHERE name = 'work_mem';
    sessionid    |       pid       |   name   | setting | unit
-----------------+-----------------+----------+---------+------
 140550214145792 |           96974 | work_mem | 65536   | kB
 140550214145792 |           96971 | work_mem | 65536   | kB
 140549731735296 | 140549731735296 | work_mem | 65536   | kB
 140549764413184 | 140549764413184 | work_mem | 65536   | kB
(4 rows)
```

- gs_local_wal_preparse_statistics()

  Description: Queries the latest startup of the log pre-parsing thread on the local node as well as the pre-parsing logs. Only the user with the SYSADMIN permission can execute this function.

  Return type: SETOF record

Name	Type	Description
preparser_term	text	Maximum term value obtained from the latest pre-parsing log.
preparser_start_time	timestamptz	Time when the latest pre-parsing is started.
preparser_end_time	timestamptz	End time of the latest pre-parsing.
preparser_start_location	text	Start position of the latest pre-parsing log.
preparser_end_location	text	End position of the latest pre-parsing log.
preparser_total_bytes	int8	Number of latest pre-parsed logs, in bytes.
preparser_speed	int8	Latest pre-parsing speed, in bytes/ms.
is_valid	bool	Specifies whether the latest pre-parsing result can be used for leader selection.

Example:

```
gaussdb=# SELECT * FROM gs_local_wal_preparse_statistics();
 preparser_term |     preparser_start_time      |      preparser_end_time       | preparser_start_location |
preparser_end_location | preparser_total_bytes | preparser_speed | is_valid
----------------+-------------------------------+-------------------------------+--------------------------
+------------------------+----------------------+-----------------+----------
 3107           | 2023-02-01 17:04:23.367946+08 | 2023-02-01 17:04:25.354434+08 | 00000003/
C3EEA660        | 00000004/0BE60738      |             1207394520 |      1207394520 | f
(1 row)
```

- gs_wlm_respool_cpu_info()

  Description: Displays the limit and usage of CPU resources in a resource pool.

  Return type: SETOF record

Name	Type	Description
respool_name	name	Name of the resource pool.
control_group	name	Cgroup name.
cpu_affinity	name	Value of cores bound to the CPU.
cpu_usage	integer	CPU usage of a resource pool.

Example:

```
gaussdb=# SELECT * FROM GS_WLM_RESPOOL_CPU_INFO();
 respool_name |    control_group     | cpu_affinity | cpu_usage
--------------+----------------------+--------------+-----------
 respool_cpu_2 | respool_cpu_2:Medium | 0-95         |         78
```



```
 default_pool  | DefaultClass:Medium  | 0-32         |        65
(2 rows)
```

- gs_wlm_respool_connection_info()

  Description: Displays the limit and usage of the number of connections in ta resource pool.

  Return type: SETOF record

Name	Type	Description
respool_name	name	Name of the resource pool.
max_connections	name	Maximum number of connections to a resource pool.
curr_connections	integer	Number of existing connections in the current resource pool.

  Example:

```
gaussdb=# SELECT * FROM GS_WLM_RESPOOL_CONNECTION_INFO();
 respool_name  | max_connections | curr_connections
---------------+-----------------+------------------
 respool1      |              -1 |                0
 default_pool  |              -1 |                1
(2 rows)
```

- gs_wlm_respool_memory_info()

  Description: Displays the limit and usage of memory resources in a resource pool.

  Return type: SETOF record

Name	Type	Description
respool_name	name	Name of the resource pool.
max_dynamic_memory	integer	Maximum dynamic memory that can be used.
current_dynamic_memory	integer	Used dynamic memory.
max_shared_memory	integer	Maximum shared memory that can be used.
current_shared_memory	integer	Shared memory that has been used.
shared_memory_hits_percent	integer	Cache hit ratio of the current resource pool.

> **NOTE**
>
> When the dynamic memory usage of the resource pool exceeds the maximum value, the returned value of the GUC parameter **current_dynamic_memory** may be greater than the value of **max_dynamic_memory**. This is normal because no memory is allocated.

Example:

```
gaussdb=# SELECT * FROM GS_WLM_RESPOOL_MEMORY_INFO();
 respool_name   | max_dynamic_memory | current_dynamic_memory | max_shared_memory |
current_shared_memory | shared_memory_hits_percent
----------------+--------------------+------------------------+-------------------+------------------------
+----------------------------
 default_pool   | -1                 | 3383kB                 | -1                | 3848kB                 |
90
 resource_pool_a | 30720kB           | 0kB                    | -1                | 0kB
|                       0
(2 rows)
```

- gs_wlm_respool_concurrency_info()

  Description: Displays the limit and usage of concurrent resources in a resource pool.

  Return type: SETOF record

Name	Type	Description
respool_name	name	Name of the resource pool.
max_concurrency	integer	Maximum number of concurrent queries allowed by the resource pool.
running_concurrency	integer	Number of concurrent tasks that are being executed in the current resource pool.
waiting_concurrency	integer	Number of concurrent tasks that are waiting in the current resource pool.

  Example:

```
gaussdb=# SELECT * FROM GS_WLM_RESPOOL_CONCURRENCY_INFO();
 respool_name   | max_concurrency | running_concurrency | waiting_concurrency
----------------+-----------------+---------------------+---------------------
 default_pool   |              -1 |                   1 |                   0
 resource_pool_a |             -1 |                   0 |                   0
(2 rows)
```

- gs_wlm_respool_io_info()

  Description: Displays the limit and usage of I/O resources in a resource pool.

  Return type: SETOF record

Name	Type	Description
respool_name	name	Name of the resource pool.

Name	Type	Description
io_limits	integer	Upper limit of IOPS. The value **0** indicates there is no limit. ● Unit: The unit is determined by GUC parameter **io_control_unit**, which is used to count the number of I/Os during I/O control. The value of **io_control_unit** is one IOPS.
io_priority	text	I/O priority set for jobs that consume I/O resources. It takes effect when the I/O usage reaches 90%. **None** indicates there is no control.
current_iops	integer	Number of times that the current I/O has been triggered. The current I/O statistics occasionally exceed the upper limit, which is related to the I/O statistics algorithm and is normal.

Example:

```
gaussdb=# SELECT * FROM GS_WLM_RESPOOL_IO_INFO();
 respool_name   | io_limits | io_priority | current_iops
----------------+-----------+-------------+--------------
 default_pool   |         0 | None        |            0
 resource_pool_a |         0 | Low         |            0
(2 rows)
```

- gs_wlm_user_space_info()

  Description: Displays the storage space usage of a user.

  Return type: SETOF record

Name	Type	Description
user_name	name	Username.
max_permanent_space	bigint	Maximum permanent storage space that can be used by a user, in bytes.
current_ permanent_space	bigint	Permanent storage space used by the current user, in bytes.
max_temp_space	bigint	Maximum temporary storage space that can be used by a user, in bytes.
current_ temp_space	bigint	Temporary storage space used by the current user, in bytes.
max_spill_space	bigint	Maximum operator flushing storage space that can be used by a user, in bytes.
current_ spill_space	bigint	Operator flushing storage space used by the current user, in bytes.

Example:

```
gaussdb=# SELECT * FROM GS_WLM_USER_SPACE_INFO();
          user_name          | max_permanent_space | current_permanent_space | max_temp_space |
current_temp_space | max_spill_space | current_spill_space
-----------------------------+---------------------+-------------------------+----------------+--------------------
+-----------------+---------------------
 xy                          |                  -1 |                    2464 |             -1 |                  0 |              -1
|                 0
(1 rows)
```

- gs_wlm_session_io_info()

  Description: Displays the I/O usage of a session.

  Return type: SETOF record

Name	Type	Description
session_id	integer	Session ID.
io_limits	integer	Upper limit of IOPS. The value **0** indicates there is no limit. ● Unit: The unit is determined by GUC parameter **io_control_unit**, which is used to count the number of I/Os during I/O control. The value of **io_control_unit** is one IOPS.
io_priority	text	I/O priority set for jobs that consume I/O resources. It takes effect when the I/O usage reaches 90%. **None** indicates there is no control.
current_iops	integer	Number of times that the current I/O has been triggered. The current I/O statistics occasionally exceed the upper limit, which is related to the I/O statistics algorithm and is normal.
wait_time	integer	Total waiting time after the current I/O exceeds the upper limit.

Example:

```
gaussdb=# SELECT * FROM GS_WLM_SESSION_IO_INFO();
   session_id    | io_limits | io_priority | current_iops | wait_time
-----------------+-----------+-------------+--------------+-----------
 139976325986048 |        10 | None        |            0 |      2709
(1 row)
```

- gs_wlm_session_memory_info()

  Description: Displays the memory usage of a session.

  Return type: SETOF record

Name	Type	Description
session_id	integer	Session ID.

Name	Type	Description
sess_used_dynamic_memory	bigint	Used dynamic memory.
sess_max_dynamic_memory	bigint	Maximum dynamic memory that can be used.

Example:

```
gaussdb=# SELECT * FROM GS_WLM_SESSION_MEMORY_INFO();
    sessid      | sess_used_dynamic_memory | sess_max_dynamic_memory
----------------+--------------------------+-------------------------
 139976325986048 |                  4326056 |                      -1
 139976402532096 |                  4452664 |                      -1
(2 rows)
```

- gs_hot_standby_space_info()

  Description: Queries the total number and total size of files in the **standby_read/base_page**, **standby_read/block_info_meta** and **standby_read/lsn_info_meta** folders.

  Return type: SETOF record

Name	Type	Description
base_page_file_num	xid	Total number of bage_page_files.
base_page_total_size	xid	Total size of bage_page_files.
lsn_info_meta_file_num	xid	Total number of lsn_info_meta_files.
lsn_info_meta_total_size	xid	Total size of the lsn_info_meta_files.
block_info_meta_file_num	xid	Total number of block_info_meta_files.
block_info_meta_total_size	xid	Total size of the block_info_meta_files.

  Example:

```
gaussdb=# SELECT * FROM gs_hot_standby_space_info();
 base_page_file_num | base_page_total_size | lsn_info_meta_file_num | lsn_info_meta_total_size |
block_info_meta_file_num | block_info_meta_total_size
--------------------+----------------------+------------------------+--------------------------
+--------------------------+---------------------------
                 6 |               163840 |                      6 |                     3136 |                       16
|                   147456
(1 row)
```

- exrto_file_read_stat()

  Description: Queries the number of disk access times and total access latency of new **base page files**, **lsn info meta files**, and **block info meta files** read

by the standby node. Connect to the standby DN for query. In other cases, the query result is 0.

Return type: SETOF record

Name	Type	Description
lsn_info_page_disk_read_counter	int8	Number of disk access times of **lsn info meta files**.
lsn_info_page_disk_read_dur	int8	Total latency of **lsn info meta file** access to disks.
blk_info_meta_disk_read_counter	int8	Number of disk access times of **block info meta files**.
blk_info_meta_disk_read_dur	int8	Total latency of **block info meta file** access to disks.
base_page_read_disk_counter	int8	Number of disk access times of **base page files**.
base_page_read_disk_dur	int8	Total latency of **base page file** access to disks.

Example:

```
gaussdb=# SELECT * FROM exrto_file_read_stat();
 lsn_info_page_disk_read_counter | lsn_info_page_disk_read_dur | blk_info_meta_disk_read_counter |
blk_info_meta_disk_read_dur | base_page_read_disk_counter | base_page_read_disk_dur
---------------------------------+-----------------------------+---------------------------------
+-----------------------------+-----------------------------+-------------------------
                          14987 |                       92313 |                           23879 |                      129811
|                           0 |                           0
(1 row)
```

- gs_exrto_recycle_info()

  Description: Queries the resource recycling location, including the recycling LSN of each thread, global recycling LSN, and the earliest snapshot LSN of a query thread. Connect to the standby DN for query. In other cases, the query result is 0.

  Return type: SETOF record

Name	Type	Description
page_redo_worker_thread_id	text	Reclamation LSN location of redo thread. **thread_id** indicates the redo thread ID.
global_recycle_lsn	text	LSN of global reclamation location.
exrto_snapshot_oldest_lsn	text	Earliest snapshot LSN of a query thread.

Example:

```
gaussdb=# SELECT * FROM gs_exrto_recycle_info();
            thread_id             | recycle_lsn
----------------------------------+-------------
 page_redo_worker_140148895381248 | 0/7B4552E0
 page_redo_worker_140148872312576 | 0/7B4535B8
 global_recycle_lsn               | 0/7B4535B8
 exrto_snapshot_oldest_lsn        | 0/8488E6D0
(4 rows)
```

- gs_stat_get_db_conflict_all(oid)

  Input parameter: **dbid(oid)** indicates the database OID.

  Description: Queries the number of sent replay conflict signals of different types.

  Return type: SETOF record

Name	Type	Description
conflict_all	int8	Number of sent replay conflict signals.
conflict_tablespace	int8	Number of sent replay conflict signals of the tablespace type.
conflict_lock	int8	Number of sent replay conflict signals of the **lock** type
conflict_snapshot	int8	Number of sent replay conflict signals of the snapshot type.
conflict_bufferpin	int8	Number of sent replay conflict signals of the bufferpin type.
conflict_startup_deadlock	int8	Number of sent replay conflict signals of the **startup_deadlock** type
conflict_truncate	int8	Number of sent replay conflict signals of the truncate type.
conflict_standby_query_timeout	int8	Number of sent replay conflict signals of the standby_query_timeout type.
conflict_force_recycle	int8	Number of sent replay conflict signals of the force_recycle type.

Example:

```
gaussdb=# SELECT * FROM gs_stat_get_db_conflict_all(12738);
 conflict_all | conflict_tablespace | conflict_lock | conflict_snapshot | conflict_bufferpin |
conflict_startup_deadlock | conflict_truncate | conflict_standby_query_timeout | conflict_force_recycle
--------------+---------------------+---------------+-------------------+--------------------
+---------------------------+-------------------+-------------------------------+------------------------
            0 |                 0 |             0 |                 0 |                 0 |                         0 |                 0
|                         0 |                   0
(1 row)
```

- gs_redo_stat_info()

  Description: Queries redo information, including the buffer hit ratio of the redo thread, number of unlink_rels files executed, wait event information of I/O operations generated when the redo thread reads the buffer in the

ultimate RTO scenario, and wait event information of **wal_read_from_write_buffer**. The query must be executed by connecting to the standby DN.

Return type: SETOF record

Name	Type	Description
buffer_hit_rate	float8	Buffer hit ratio of the redo thread.
ddl_unlink_nrels_count	int8	Number of unlink rel files executed during the redo process of DDL statements.
read_buffer_io_counter	int8	Number of wait events of I/O operations generated when the redo thread reads the buffer in the ultimate RTO scenario.
read_buffer_io_total_dur	int8	Total wait event duration of I/O operations generated when the redo thread reads the buffer in the ultimate RTO scenario.
read_buffer_io_avg_dur	int8	Average wait event duration of I/O operations generated when the redo thread reads the buffer in the ultimate RTO scenario.
read_buffer_io_min_dur	int8	Minimum wait event duration of I/O operations generated when the redo thread reads the buffer in the ultimate RTO scenario.
read_buffer_io_max_dur	int8	Maximum wait event duration of I/O operations generated when the redo thread reads the buffer in the ultimate RTO scenario.
read_wal_buf_counter	int8	Number of wait events triggered by **wal_read_from_write_buffer** in the ultimate RTO scenario.
read_wal_buf_total_dur	int8	Total wait event duration of **wal_read_from_write_buffer** in the ultimate RTO scenario.
read_wal_buf_avg_dur	int8	Average wait event duration of **wal_read_from_write_buffer** in the ultimate RTO scenario.
read_wal_buf_min_dur	int8	Minimum wait event duration of **wal_read_from_write_buffer** in the ultimate RTO scenario.
read_wal_buf_max_dur	int8	Maximum wait event duration of **wal_read_from_write_buffer** in the ultimate RTO scenario.

Example:

```
gaussdb=# SELECT * FROM gs_redo_stat_info();
-[ RECORD 1 ]------------+----------
```



```
buffer_hit_rate          | 70.5707
ddl_unlink_nrels_count   | 3
read_buffer_io_counter   | 1732
read_buffer_io_total_dur | 2850806
read_buffer_io_avg_dur   | 1645
read_buffer_io_min_dur   | 3
read_buffer_io_max_dur   | 981639
read_wal_buf_counter     | 9779
read_wal_buf_total_dur   | 193612470
read_wal_buf_avg_dur     | 19798
read_wal_buf_min_dur     | 3
read_wal_buf_max_dur     | 1914777
```

- gs_recovery_conflict_waitevent_info()

  Description: Queries wait event information about the function that processes redo conflicts. The query must be executed by connecting to the standby DN.

  Return type: SETOF record

Name	Type	Description
conflict_lock_counter	int8	Number of times that lock redo conflicts are triggered.
conflict_lock_total_dur	int8	Total duration for processing lock redo conflicts.
conflict_lock_avg_dur	int8	Average duration for processing lock redo conflicts.
conflict_lock_min_dur	int8	Minimum duration for processing lock redo conflicts.
conflict_lock_max_dur	int8	Maximum duration for processing lock redo conflicts.
conflict_snapshot_counter	int8	Number of times that snapshot redo conflicts are triggered.
conflict_snapshot_total_dur	int8	Total duration for processing snapshot redo conflicts.
conflict_snapshot_avg_dur	int8	Average duration for processing snapshot redo conflicts.
conflict_snapshot_min_dur	int8	Minimum duration for processing snapshot redo conflicts.
conflict_snapshot_max_dur	int8	Maximum duration for processing snapshot redo conflicts.
conflict_tablespace_counter	int8	Number of times that tablespace redo conflicts are triggered.
conflict_tablespace_total_dur	int8	Total duration for processing tablespace redo conflicts.
conflict_tablespace_avg_dur	int8	Average duration for processing tablespace redo conflicts.

Name	Type	Description
conflict_tablespace_min_dur	int8	Minimum duration for processing tablespace redo conflicts.
conflict_tablespace_max_dur	int8	Maximum duration for processing tablespace redo conflicts.
conflict_database_counter	int8	Number of times that database redo conflicts are triggered.
conflict_database_total_dur	int8	Total duration for processing database redo conflicts.
conflict_database_avg_dur	int8	Average duration for processing database redo conflicts.
conflict_database_min_dur	int8	Minimum duration for processing database redo conflicts.
conflict_database_max_dur	int8	Maximum duration for processing database redo conflicts.
conflict_truncate_counter	int8	Number of times that TRUNCATE redo conflicts are triggered.
conflict_truncate_total_dur	int8	Total duration for processing TRUNCATE redo conflicts.
conflict_truncate_avg_dur	int8	Average duration for processing TRUNCATE redo conflicts.
conflict_truncate_min_dur	int8	Minimum duration for processing TRUNCATE redo conflicts.
conflict_truncate_max_dur	int8	Maximum duration for processing TRUNCATE redo conflicts.
conflict_standby_query_timeout_counter	int8	Number of times that standby_query_timeout redo conflicts are triggered.
conflict_standby_query_timeout_total_dur	int8	Total duration for processing standby_query_timeout redo conflicts.
conflict_standby_query_timeout_avg_dur	int8	Average duration for processing standby_query_timeout redo conflicts.
conflict_standby_query_timeout_min_dur	int8	Minimum duration for processing standby_query_timeout redo conflicts.
conflict_standby_query_timeout_max_dur	int8	Maximum duration for processing standby_query_timeout redo conflicts.

Name	Type	Description
conflict_force_recycle_counter	int8	Number of times that force_recycle redo conflicts are triggered.
conflict_force_recycle_total_dur	int8	Total duration for processing force_recycle redo conflicts.
conflict_force_recycle_avg_dur	int8	Average duration for processing force_recycle redo conflicts.
conflict_force_recycle_min_dur	int8	Minimum duration for processing force_recycle redo conflicts.
conflict_force_recycle_max_dur	int8	Maximum duration for processing force_recycle redo conflicts.

Example:

```
gaussdb=# SELECT * FROM gs_recovery_conflict_waitevent_info();
-[ RECORD 1 ]----------------------------+------
conflict_lock_counter                    | 0
conflict_lock_total_dur                  | 0
conflict_lock_avg_dur                    | 0
conflict_lock_min_dur                    | 0
conflict_lock_max_dur                    | 0
conflict_snapshot_counter                | 0
conflict_snapshot_total_dur              | 0
conflict_snapshot_avg_dur                | 0
conflict_snapshot_min_dur                | 0
conflict_snapshot_max_dur                | 0
conflict_tablespace_counter              | 0
conflict_tablespace_total_dur            | 0
conflict_tablespace_avg_dur              | 0
conflict_tablespace_min_dur              | 0
conflict_tablespace_max_dur              | 0
conflict_database_counter                | 0
conflict_database_total_dur              | 0
conflict_database_avg_dur                | 0
conflict_database_min_dur                | 0
conflict_database_max_dur                | 0
conflict_truncate_counter                | 6
conflict_truncate_total_dur              | 35872
conflict_truncate_avg_dur                | 5978
conflict_truncate_min_dur                | 5130
conflict_truncate_max_dur                | 7459
conflict_standby_query_timeout_counter   | 0
conflict_standby_query_timeout_total_dur | 0
conflict_standby_query_timeout_avg_dur   | 0
conflict_standby_query_timeout_min_dur   | 0
conflict_standby_query_timeoutmax_dur    | 0
conflict_force_recycle_counter           | 0
conflict_force_recycle_total_dur         | 0
conflict_force_recycle_avg_dur           | 0
conflict_force_recycle_min_dur           | 0
conflict_force_recycle_max_dur           | 0
```

- gs_display_delay_ddl_info()

  Description: Views information about files that are delayed for deletion on the standby node.

  Return type: SETOF record

Table 7-147

Name	Type	Description
type	INT4	Indicates that the deleted object is a table or database.
lsn	TEXT	Marks the location of a particular log file.
tablespace	INT4	Indicates the physical space for storing tables and indexes in a database.
database	INT4	Indicates the physical storage location of a database.
relation	INT4	Indicates the object in a database, which can be the physical location of a table, view, or index.
bucketid	INT4	Specifies the bucket to which the relationship object belongs.
opt	INT4	Indicates the attribute of a compressed table.
forknum	INT4	Specifies a suffix name for a subject name. You can find a unique physical file based on the subject name and suffix name.

Example:

```
gaussdb=# SELECT * FROM gs_display_delay_ddl_info();
 type | lsn | tablespace | database | relation | bucketid | opt | forknum
------+-----+------------+----------+----------+----------+-----+---------
(0 rows)
```

Functions for Collecting Statistics in Partitioned Tables

- gs_stat_get_partition_stats(oid)
  Description: Obtains the statistics of a specific partition.
  Return type: record
- gs_stat_get_xact_partition_stats(oid)
  Description: Obtains transaction statistics of a specific partition.
  Return type: record
- gs_stat_get_all_partitions_stats()
  Description: Obtains the statistics of all partitions.
  Return type: setof record
- gs_stat_get_xact_all_partitions_stats()
  Description: Obtains transaction statistics of all partitions.

Return type: setof record

- gs_statio_get_all_partitions_stats()

  Description: Obtains the I/O statistics of all partitions.

  Return type: setof record

  Examples of the preceding five functions

> **CAUTION**
>
> Statistics are reported asynchronously during execution. Based on UDP, delay and packet loss may occur during background thread processing. The following example is for reference only.

Querying out-of-transaction statistics:

```
gaussdb=# CREATE TABLE part_tab1
gaussdb-# (
gaussdb(#     a int, b int
gaussdb(# )
gaussdb-# PARTITION BY RANGE(b)
gaussdb-# (
gaussdb(#     PARTITION P1 VALUES LESS THAN(10),
gaussdb(#     PARTITION P2 VALUES LESS THAN(20),
gaussdb(#     PARTITION P3 VALUES LESS THAN(MAXVALUE)
gaussdb(# );
CREATE TABLE
gaussdb=# CREATE TABLE subpart_tab1
gaussdb-# (
gaussdb(#     month_code VARCHAR2 ( 30 ) NOT NULL ,
gaussdb(#     dept_code  VARCHAR2 ( 30 ) NOT NULL ,
gaussdb(#     user_no    VARCHAR2 ( 30 ) NOT NULL ,
gaussdb(#     sales_amt  int
gaussdb(# )
gaussdb-# PARTITION BY RANGE (month_code) SUBPARTITION BY RANGE (dept_code)
gaussdb-# (
gaussdb(#   PARTITION p_201901 VALUES LESS THAN( '201903' )
gaussdb(#   (
gaussdb(#     SUBPARTITION p_201901_a VALUES LESS THAN( '2' ),
gaussdb(#     SUBPARTITION p_201901_b VALUES LESS THAN( '3' )
gaussdb(#   ),
gaussdb(#   PARTITION p_201902 VALUES LESS THAN( '201904' )
gaussdb(#   (
gaussdb(#     SUBPARTITION p_201902_a VALUES LESS THAN( '2' ),
gaussdb(#     SUBPARTITION p_201902_b VALUES LESS THAN( '3' )
gaussdb(#   )
gaussdb(# );
CREATE TABLE
gaussdb=# CREATE INDEX index_part_tab1 ON part_tab1(b) LOCAL
gaussdb-# (
gaussdb(# PARTITION b_index1,
gaussdb(# PARTITION b_index2,
gaussdb(# PARTITION b_index3
gaussdb(# );
CREATE INDEX
gaussdb=# CREATE INDEX idx_user_no ON subpart_tab1(user_no) LOCAL;
CREATE INDEX
gaussdb=# INSERT INTO part_tab1 VALUES(1, 1);
INSERT 0 1
gaussdb=# INSERT INTO part_tab1 VALUES(1, 11);
INSERT 0 1
gaussdb=# INSERT INTO part_tab1 VALUES(1, 21);
INSERT 0 1
gaussdb=# UPDATE part_tab1 SET a = 2 WHERE b = 1;
UPDATE 1
```



```
gaussdb=# UPDATE part_tab1 SET a = 3 WHERE b = 11;
UPDATE 1
gaussdb=# UPDATE /*+ indexscan(part_tab1) */ part_tab1 SET a = 4 WHERE b = 21;
UPDATE 1
gaussdb=# DELETE FROM part_tab1;
DELETE 3
gaussdb=# ANALYZE part_tab1;
ANALYZE
gaussdb=# VACUUM part_tab1;
VACUUM
gaussdb=# INSERT INTO subpart_tab1 VALUES('201902', '1', '1', 1);
INSERT 0 1
gaussdb=# INSERT INTO subpart_tab1 VALUES('201902', '2', '2', 1);
INSERT 0 1
gaussdb=# INSERT INTO subpart_tab1 VALUES('201903', '1', '3', 1);
INSERT 0 1
gaussdb=# INSERT INTO subpart_tab1 VALUES('201903', '2', '4', 1);
INSERT 0 1
gaussdb=# UPDATE subpart_tab1 SET sales_amt = 2 WHERE user_no='1';
UPDATE 1
gaussdb=# UPDATE subpart_tab1 SET sales_amt = 3 WHERE user_no='2';
UPDATE 1
gaussdb=# UPDATE subpart_tab1 SET sales_amt = 4 WHERE user_no='3';
UPDATE 1
gaussdb=# UPDATE /*+ indexscan(subpart_tab1) */ subpart_tab1 SET sales_amt = 5 WHERE
user_no='4';
UPDATE 1
gaussdb=# DELETE FROM subpart_tab1;
DELETE 4
gaussdb=# ANALYZE subpart_tab1;
ANALYZE
gaussdb=# VACUUM subpart_tab1;
VACUUM
gaussdb=# SELECT * FROM gs_stat_all_partitions;
 partition_oid | schemaname |    relname     | partition_name | sub_partition_name | seq_scan |
seq_tup_read | idx_scan | idx_tup_fetch | n_tup_ins | n_tup_upd | n_tup_del | n_tup_hot_upd | n_live_tup
|
 n_dead_tup |          last_vacuum          |     last_autovacuum     |         last_analyze          |
last_autoanalyze     | vacuum_count | autovacuum_count | analyze_count | autoanalyze_count
---------------+------------+-------------+---------------+--------------------+----------+-------------
+----------+---------------+----------+----------+----------+---------------+------------+
------------+-------------------------------+-----------------------+-------------------------------
+-----------------------+--------------+-----------------+---------------+-------------------
        16964 | public     | subpart_tab1 | p_201902       | p_201902_b         |        5 |          1 |       4
|           1 |         1 |         1 |         1 |             1 |          0 |
         1 | 2023-05-15 20:36:45.293965+08 | 2000-01-01 08:00:00+08 | 2023-05-15
20:36:44.688861+08 | 2000-01-01 08:00:00+08 |           1 |              0 |            1 |              0
        16963 | public     | subpart_tab1 | p_201902       | p_201902_a         |        5 |          1 |       4
|           0 |         1 |         1 |         1 |             1 |          0 |
         1 | 2023-05-15 20:36:45.291022+08 | 2000-01-01 08:00:00+08 | 2023-05-15
20:36:44.688843+08 | 2000-01-01 08:00:00+08 |           1 |              0 |            1 |              0
        16961 | public     | subpart_tab1 | p_201901       | p_201901_b         |        5 |          1 |       4
|           0 |         1 |         1 |         1 |             1 |          0 |
         1 | 2023-05-15 20:36:45.288037+08 | 2000-01-01 08:00:00+08 | 2023-05-15
20:36:44.688829+08 | 2000-01-01 08:00:00+08 |           1 |              0 |            1 |              0
        16960 | public     | subpart_tab1 | p_201901       | p_201901_a         |        5 |          1 |       4
|           0 |         1 |         1 |         1 |             1 |          0 |
         1 | 2023-05-15 20:36:45.285311+08 | 2000-01-01 08:00:00+08 | 2023-05-15
20:36:44.688802+08 | 2000-01-01 08:00:00+08 |           1 |              0 |            1 |              0
        16954 | public     | part_tab1    | p3             |                    |        2 |          1 |       1 |           1
|           1 |         1 |         1 |         1 |             1 |          0 |
         1 | 2023-05-15 20:36:29.490636+08 | 2000-01-01 08:00:00+08 | 2023-05-15
20:36:28.540115+08 | 2000-01-01 08:00:00+08 |           1 |              0 |            1 |              0
        16953 | public     | part_tab1    | p2             |                    |        4 |          1 |       1 |           0
|           1 |         1 |         1 |         1 |             1 |          0 |
         1 | 2023-05-15 20:36:29.487914+08 | 2000-01-01 08:00:00+08 | 2023-05-15
20:36:28.540098+08 | 2000-01-01 08:00:00+08 |           1 |              0 |            1 |              0
        16952 | public     | part_tab1    | p1             |                    |        5 |          1 |       1 |           0
|           1 |         1 |         1 |         1 |             1 |          0 |
```



```
         1 | 2023-05-15 20:36:29.48536+08  | 2000-01-01 08:00:00+08 | 2023-05-15 20:36:28.540071+08
| 2000-01-01 08:00:00+08 |            1 |               0 |            1 |               0
(7 rows)

gaussdb=# SELECT * FROM gs_statio_all_partitions;
 partition_oid | schemaname |    relname     | partition_name | sub_partition_name | heap_blks_read |
heap_blks_hit | idx_blks_read | idx_blks_hit | toast_blks_read | toast_blks_hit | tidx_blks_read | t
idx_blks_hit
---------------+------------+--------------+----------------+--------------------+----------------+--------------
+---------------+--------------+-----------------+----------------+----------------+--
-------------
        16964 | public     | subpart_tab1 | p_201902       | p_201902_b         |              4 |            8
|             2 |           21 |                 |                |                |

        16963 | public     | subpart_tab1 | p_201902       | p_201902_a         |              4 |            8
|             2 |           21 |                 |                |                |

        16961 | public     | subpart_tab1 | p_201901       | p_201901_b         |              4 |            8
|             2 |           21 |                 |                |                |

        16960 | public     | subpart_tab1 | p_201901       | p_201901_a         |              4 |            8
|             2 |           21 |                 |                |                |

        16954 | public     | part_tab1    | p3             |                    |              4 |            8 |            2
|          15 |              |                 |                |

        16953 | public     | part_tab1    | p2             |                    |              4 |            8 |            2
|          15 |              |                 |                |

        16952 | public     | part_tab1    | p1             |                    |              4 |            8 |            2
|          15 |              |                 |                |

(7 rows)

gaussdb=# SELECT * FROM gs_stat_get_partition_stats(16952);
 partition_oid | seq_scan | seq_tup_read | idx_scan | idx_tup_fetch | n_tup_ins | n_tup_upd | n_tup_del |
n_tup_hot_upd | n_live_tup | n_dead_tup |          last_vacuum          |     last_autovacuum
  |         last_analyze          |     last_autoanalyze     | vacuum_count | autovacuum_count |
analyze_count | autoanalyze_count |    last_data_changed     | heap_blks_read | heap_blks_hit |
idx_blks_re
ad | idx_blks_hit | tup_fetch | block_fetch
---------------+----------+--------------+----------+---------------+-----------+-----------+-----------
+---------------+------------+------------+-------------------------------+----------------------
--+-------------------------------+-----------------------+-------------+-----------------+---------------
+-------------------+-----------------------+----------------+---------------+------------
---+--------------+-----------+-------------
        16952 |        5 |            1 |        1 |             0 |         1 |         1 |         1 |             1 |          0
|          1 | 2023-05-15 20:36:29.48536+08 | 2000-01-01 08:00:00+0
8 | 2023-05-15 20:36:28.540071+08 | 2000-01-01 08:00:00+08 |             1 |               0 |             1
|                0 | 2000-01-01 08:00:00+08 |              4 |             8 |
 2 |           21 |         0 |          12
(1 row)
```

Querying statistics within a transaction:

```
gaussdb=# BEGIN;
BEGIN
gaussdb=# INSERT INTO part_tab1 VALUES(1, 1);
INSERT 0 1
gaussdb=# INSERT INTO part_tab1 VALUES(1, 11);
INSERT 0 1
gaussdb=# INSERT INTO part_tab1 VALUES(1, 21);
INSERT 0 1
gaussdb=# UPDATE part_tab1 SET a = 2 WHERE b = 1;
UPDATE 1
gaussdb=# UPDATE part_tab1 SET a = 3 WHERE b = 11;
UPDATE 1
gaussdb=# UPDATE /*+ indexscan(part_tab1) */ part_tab1 SET a = 4 WHERE b = 21;
UPDATE 1
gaussdb=# DELETE FROM part_tab1;
DELETE 3
```



```
gaussdb=# INSERT INTO subpart_tab1 VALUES('201902', '1', '1', 1);
INSERT 0 1
gaussdb=# INSERT INTO subpart_tab1 VALUES('201902', '2', '2', 1);
INSERT 0 1
gaussdb=# INSERT INTO subpart_tab1 VALUES('201903', '1', '3', 1);
INSERT 0 1
gaussdb=# INSERT INTO subpart_tab1 VALUES('201903', '2', '4', 1);
INSERT 0 1
gaussdb=# UPDATE subpart_tab1 SET sales_amt = 2 WHERE user_no='1';
UPDATE 1
gaussdb=# UPDATE subpart_tab1 SET sales_amt = 3 WHERE user_no='2';
UPDATE 1
gaussdb=# UPDATE subpart_tab1 SET sales_amt = 4 WHERE user_no='3';
UPDATE 1
gaussdb=# UPDATE /*+ indexscan(subpart_tab1) */ subpart_tab1 SET sales_amt = 5 WHERE
user_no='4';
UPDATE 1
gaussdb=# DELETE FROM subpart_tab1;
DELETE 4
gaussdb=# SELECT * FROM gs_stat_xact_all_partitions;
 partition_oid | schemaname |    relname     | partition_name | sub_partition_name | seq_scan |
seq_tup_read | idx_scan | idx_tup_fetch | n_tup_ins | n_tup_upd | n_tup_del | n_tup_hot_upd
---------------+------------+----------------+----------------+--------------------+----------+--------------
+----------+---------------+-----------+-----------+-----------+---------------
        16964 | public     | subpart_tab1 | p_201902       | p_201902_b         |        4 |           4 |        1
|           2 |         1 |         1 |         1 |             1
        16963 | public     | subpart_tab1 | p_201902       | p_201902_a         |        4 |           4 |        1
|           0 |         1 |         1 |         1 |             1
        16961 | public     | subpart_tab1 | p_201901       | p_201901_b         |        4 |           4 |        1
|           0 |         1 |         1 |         1 |             1
        16960 | public     | subpart_tab1 | p_201901       | p_201901_a         |        4 |           4 |        1
|           0 |         1 |         1 |         1 |             1
        16954 | public     | part_tab1    | p3             |                    |        1 |           1 |        1 |           2
|         1 |         1 |         1 |             1
        16953 | public     | part_tab1    | p2             |                    |        3 |           2 |        0 |           0
|         1 |         1 |         1 |             1
        16952 | public     | part_tab1    | p1             |                    |        4 |           2 |        0 |           0
|         1 |         1 |         1 |             1
(7 rows)

gaussdb=# SELECT * FROM gs_stat_get_xact_partition_stats(16952);
 partition_oid | seq_scan | seq_tup_read | idx_scan | idx_tup_fetch | n_tup_ins | n_tup_upd | n_tup_del |
n_tup_hot_upd | tup_fetch
---------------+----------+--------------+----------+---------------+-----------+-----------+-----------
+---------------+-----------
        16952 |        4 |           2 |        0 |             0 |         1 |         1 |         1 |             1 |         0
(1 row)
```

- gs_stat_get_partition_analyze_count(oid)

  Description: Specifies the number of times that a user starts analysis on a partition.

  Return type: bigint

- gs_stat_get_partition_autoanalyze_count(oid)

  Description: Specifies the number of times that the autovacuum daemon thread starts analysis in a partition.

  Return type: bigint

- gs_stat_get_partition_autovacuum_count(oid)

  Description: Specifies the number of times that the autovacuum daemon thread starts vacuum in a partition.

  Return type: bigint

- gs_stat_get_partition_last_analyze_time(oid)

Description: Specifies the last time when a partition starts to be analyzed manually or by the autovacuum thread.

Return type: timestamptz

- gs_stat_get_partition_last_autoanalyze_time(oid)

  Description: Specifies the time when the last analysis initiated by the autovacuum daemon thread in a partition.

  Return type: timestamptz

- gs_stat_get_partition_last_autovacuum_time(oid)

  Description: Specifies the time of the last vacuum initiated by the autovacuum daemon thread in a partition.

  Return type: timestamptz

- gs_stat_get_partition_last_data_changed_time(oid)

  Description: Specifies the last time of a modification in a partition, such as insert, update, delete, and truncate. Currently, this parameter is not supported.

  Return type: timestamptz

- gs_stat_get_partition_last_vacuum_time(oid)

  Description: Specifies the most recent time when the user manually cleared a table or when the autovacuum thread was started to clear a partition.

  Return type: timestamptz

- gs_stat_get_partition_numscans(oid)

  Description: Specifies the number of rows scanned and read in partition order.

  Return type: bigint

- gs_stat_get_partition_tuples_returned(oid)

  Description: Specifies the number of rows scanned and read in partition order.

  Return type: bigint

- gs_stat_get_partition_tuples_fetched(oid)

  Description: Specifies the number of rows fetched by bitmap scans in a partition.

  Return type: bigint

- gs_stat_get_partition_vacuum_count(oid)

  Description: Specifies the number of times that a user starts vacuum in a partition.

  Return type: bigint

- gs_stat_get_xact_partition_tuples_fetched(oid)

  Description: Specifies the number of tuple rows scanned in a transaction.

  Return type: bigint

- gs_stat_get_xact_partition_numscans(oid)

  Description: Specifies the number of sequential scans performed on a partition in the current transaction.

  Return type: bigint

- gs_stat_get_xact_partition_tuples_returned(oid)

Description: Specifies the number of rows read through sequential scans in a partition in the current transaction.

Return type: bigint

- gs_stat_get_partition_blocks_fetched(oid)

  Description: Specifies the number of disk block fetch requests for a partition.

  Return type: bigint

- gs_stat_get_partition_blocks_hit(oid)

  Description: Specifies the number of disk block requests found in cache for a partition.

  Return type: bigint

- pg_stat_get_partition_tuples_inserted(oid)

  Description: Specifies the number of rows in the corresponding table partition.

  Return type: bigint

- pg_stat_get_partition_tuples_updated(oid)

  Description: Specifies the number of rows that have been updated in the corresponding table partition.

  Return type: bigint

- pg_stat_get_partition_tuples_deleted(oid)

  Description: Specifies the number of rows deleted from the corresponding table partition.

  Return type: bigint

- pg_stat_get_partition_tuples_changed(oid)

  Description: Specifies the total number of inserted, updated, and deleted rows after a table partition was last analyzed or autoanalyzed.

  Return type: bigint

- pg_stat_get_partition_live_tuples(oid)

  Description: Specifies the number of live rows in a partitioned table.

  Return type: bigint

- pg_stat_get_partition_dead_tuples(oid)

  Description: Specifies the number of dead rows in a partitioned table.

  Return type: bigint

- pg_stat_get_xact_partition_tuples_inserted(oid)

  Description: Specifies the number of inserted tuples in the active sub-transactions related to a table partition.

  Return type: bigint

- pg_stat_get_xact_partition_tuples_deleted(oid)

  Description: Specifies the number of deleted tuples in the active sub-transactions related to a table partition.

  Return type: bigint

- pg_stat_get_xact_partition_tuples_hot_updated(oid)

  Description: Specifies the number of hot updated tuples in the active sub-transactions related to a table partition.

  Return type: bigint

- pg_stat_get_xact_partition_tuples_updated(oid)

  Description: Specifies the number of updated tuples in the active sub-transactions related to a table partition.

  Return type: bigint

- pg_stat_get_partition_tuples_hot_updated(oid)

  Description: Returns statistics on the number of hot updated tuples in a partition with a specified partition ID.

  Parameter: **oid**

  Return type: bigint

7.6.29 Trigger Functions

- pg_get_triggerdef(oid)

  Description: Obtains the definition information of a trigger.

  Parameter: OID of the trigger to be queried

  Return type: text

  Example:

```
-- Create the tri_insert table.
gaussdb=# CREATE TABLE tri_insert (a int, b int);
CREATE TABLE
-- Create the trigger_func function.
gaussdb=# CREATE FUNCTION trigger_func() RETURNS trigger LANGUAGE plpgsql AS '
gaussdb'# BEGIN
gaussdb'# RAISE NOTICE ''trigger_func(%) called: action = %, when = %, level = %'', TG_ARGV[0],
TG_OP, TG_WHEN, TG_LEVEL;
gaussdb'# RETURN NULL;
gaussdb'# END;';
CREATE FUNCTION
-- Create the before_ins_stmt_trig trigger.
gaussdb=# CREATE TRIGGER before_ins_stmt_trig BEFORE INSERT ON tri_insert
gaussdb-# FOR EACH STATEMENT EXECUTE PROCEDURE trigger_func('before_ins_stmt');
CREATE TRIGGER
-- Create the after_ins_when_trig trigger.
gaussdb=# CREATE TRIGGER after_ins_when_trig AFTER INSERT ON tri_insert
gaussdb-# FOR EACH ROW WHEN (new.a IS NOT NULL) EXECUTE PROCEDURE
trigger_func('after_ins_when');
CREATE TRIGGER
-- View the trigger definition information of the tri_insert table.
gaussdb=# SELECT pg_get_triggerdef(oid) FROM pg_trigger WHERE tgrelid = 'tri_insert'::regclass;

pg_get_triggerdef
-----------------------------------------------------------------------------------------------------------------------------------
 CREATE TRIGGER before_ins_stmt_trig BEFORE INSERT ON tri_insert FOR EACH STATEMENT
EXECUTE PROCEDURE trigger_func('before_ins_stmt')
 CREATE TRIGGER after_ins_when_trig AFTER INSERT ON tri_insert FOR EACH ROW WHEN ((new.a IS
NOT NULL)) EXECUTE PROCEDURE trigger_func('after_ins_when')
(2 rows)
```

- pg_get_triggerdef(oid, boolean)

  Description: Obtains the definition information of a trigger.

  Parameter: OID of the trigger to be queried and whether it is displayed in pretty mode

  > **NOTE**
  >
  > Boolean parameters take effect only when the WHEN condition is specified during trigger creation.

Return type: text

Example:

```
-- View the trigger definition information of the tri_insert table in non-pretty mode.
gaussdb=# SELECT pg_get_triggerdef(oid, false) FROM pg_trigger WHERE tgrelid = 'tri_insert'::regclass;

pg_get_triggerdef
------------------------------------------------------------------------------------------------------------------------------------------------------------
 CREATE TRIGGER before_ins_stmt_trig BEFORE INSERT ON tri_insert FOR EACH STATEMENT EXECUTE PROCEDURE trigger_func('before_ins_stmt')
 CREATE TRIGGER after_ins_when_trig AFTER INSERT ON tri_insert FOR EACH ROW WHEN ((new.a IS NOT NULL)) EXECUTE PROCEDURE trigger_func('after_ins_when')
(2 rows)

-- View the trigger definition information of the tri_insert table in pretty mode.
gaussdb=# SELECT pg_get_triggerdef(oid, true) FROM pg_trigger WHERE tgrelid = 'tri_insert'::regclass;

pg_get_triggerdef
------------------------------------------------------------------------------------------------------------------------------------------------------------
 CREATE TRIGGER before_ins_stmt_trig BEFORE INSERT ON tri_insert FOR EACH STATEMENT EXECUTE PROCEDURE trigger_func('before_ins_stmt')
 CREATE TRIGGER after_ins_when_trig AFTER INSERT ON tri_insert FOR EACH ROW WHEN (new.a IS NOT NULL) EXECUTE PROCEDURE trigger_func('after_ins_when')
(2 rows)
-- Clear the tri_insert table.
gaussdb=# DROP TABLE tri_insert CASCADE;
DROP TABLE
-- Clear the trigger_func function.
gaussdb=# DROP FUNCTION trigger_func;
DROP FUNCTION
```

7.6.30 Hash Functions

- ora_hash(expression,[seed])

  Description: Calculates the hash value of a given expression. **expression**: The value can be a character string, time, or number. The hash value is calculated based on the expression. **seed**: an int8 value that can return different results for the same input value. This parameter is optional and is used to calculate the hash value with a random number.

  Return type: hash value of the int8 type.

  Example:

```
gaussdb=# SELECT ora_hash(123);
 ora_hash
------------
 4089882933
(1 row)
gaussdb=# SELECT ora_hash('123');
 ora_hash
------------
 2034089965
(1 row)
gaussdb=# SELECT ora_hash('sample');
 ora_hash
------------
 1573005290
(1 row)
gaussdb=# SELECT ora_hash(to_date('2012-1-2','yyyy-mm-dd'));
 ora_hash
------------
 1171473495
(1 row)
gaussdb=# SELECT ora_hash(123,234);
```



```
 ora_hash
------------
 -9089505052966355682
(1 row)
gaussdb=# SELECT ora_hash('123',234);
 ora_hash
------------
 5742589019960764616
(1 row)
gaussdb=# SELECT ora_hash('sample',234);
 ora_hash
------------
 -1747984408055821656
(1 row)
gaussdb=# SELECT ora_hash(to_date('2012-1-2','yyyy-mm-dd'),234);
 ora_hash
------------
 -3306025179710572679
(1 row)
```

NOTE

This function is valid only when **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s2**.

- hash_array(anyarray)

  Description: Hashes an array, obtains the result of an array element using the hash function, and returns the combination result.

  Parameter: data of the anyarray type.

  Return type: integer

  Example:

  ```
  gaussdb=# SELECT hash_array(ARRAY[[1,2,3],[1,2,3]]);
   hash_array
  ------------
   -382888479
  (1 row)
  ```

- hash_numeric(numeric)

  Description: Calculates the hash value of numeric data.

  Parameter: data of the numeric type.

  Return type: integer

  Example:

  ```
  gaussdb=# SELECT hash_numeric(30);
   hash_numeric
  --------------
     -282860963
  (1 row)
  ```

- hash_range(anyrange)

  Description: Calculates the hash value of a range.

  Parameter: data of the anyrange type.

  Return type: integer

  Example:

  ```
  gaussdb=# SELECT hash_range(numrange(1.1,2.2));
   hash_range
  ------------
    683508754
  (1 row)
  ```

- hashbpchar(character)

Description: Calculates the hash value of bpchar.

Parameter: data of the character type.

Return type: integer

Example:

```
gaussdb=# SELECT hashbpchar('hello');
 hashbpchar
-------------
 -1870292951
(1 row)
```

- hashchar(char)

  Description: Converts char and Boolean data into hash values.

  Parameter: data of the char or Boolean type.

  Return type: integer

  Example:

  ```
  gaussdb=# SELECT hashbpchar('hello');
   hashbpchar
  -------------
   -1870292951
  (1 row)

  gaussdb=# SELECT hashchar('true');
    hashchar
  ------------
   1686226652
  (1 row)
  ```

- hashenum(anyenum)

  Description: Converts enumerated values to hash values.

  Parameter: data of the anyenum type.

  Return type: integer

  Example:

  ```
  gaussdb=# CREATE TYPE b1 AS ENUM('good', 'bad', 'ugly');
  CREATE TYPE
  gaussdb=# call hashenum('good'::b1);
    hashenum
  ------------
   1821213359
  (1 row)
  ```

- hashfloat4(real)

  Description: Converts float4 values to hash values.

  Parameter: data of the real type.

  Return type: integer

  Example:

  ```
  gaussdb=# SELECT hashfloat4(12.1234);
   hashfloat4
  ------------
   1398514061
  (1 row)
  ```

- hashfloat8(double precision)

  Description: Converts float8 values to hash values.

  Parameter: data of the double precision type.

  Return type: integer

  Example:


```
gaussdb=# SELECT hashfloat8(123456.1234);
 hashfloat8
------------
 1673665593
(1 row)
```

- hashinet(inet)

  Description: Converts inet or cidr values to hash values.

  Parameter: data of the inet type.

  Return type: integer

  Example:

```
gaussdb=# SELECT hashinet('127.0.0.1'::inet);
  hashinet
-------------
 -1435793109
(1 row)
```

- hashint1(tinyint)

  Description: Converts INT1 values to hash values.

  Parameter: data of the tinyint type.

  Return type: uint32

  Example:

```
gaussdb=# SELECT hashint1(20);
  hashint1
-------------
 -2014641093
(1 row)
```

- hashint2(smallint)

  Description: Converts INT2 values to hash values.

  Parameter: data of the smallint type.

  Return type: uint32

  Example:

```
gaussdb=# SELECT hashint2(20000);
  hashint2
------------
 -863179081
(1 row)
```

7.6.31 Prompt Message Function

- report_application_error

  Description: This function can be used to throw errors during PL execution.

  Return type: void

  Table 7-148 report_application_error parameter description

Parameter	Type	Description	Mandatory or Not
log	text	Content of an error message.	Yes
code	int4	Error code corresponding to an error message. The value ranges from –20999 to –20000.	No

7.6.32 Global Temporary Table Functions

- pg_get_gtt_relstats(relOid)

  Description: Displays basic information about a global temporary table specified by the current session.

  Parameter: OID of the global temporary table

  Return type: record

  Example:

```
gaussdb=# SELECT * FROM pg_get_gtt_relstats(74069);
 relfilenode | relpages | reltuples | relallvisible | relfrozenxid | relminmxid
-------------+----------+-----------+---------------+--------------+------------
       74069 |       58 |     13000 |             0 |        11151 |          0
(1 row)
```

- pg_get_gtt_statistics(relOid, attnum, ''::text)

  Description: Displays statistics about a single column in a global temporary table specified by the current session.

  Parameter: OID and the **attnum** attribute of the global temporary table

  Return type: record

  Example:

```
gaussdb=# SELECT * FROM  pg_get_gtt_statistics(74069,1,''::text);
 starelid | starelkind | staattnum | stainherit | stanullfrac | stawidth | stadistinct | stakind1 | stakind2 |
stakind3 | stakind4 | stakind5 | staop1 | staop2 | staop3 | staop4 | staop5 | stanumbers1 | stanumbers2
| stanumbers3 | stanu
mbers4 | stanumbers5
|


stavalues1

                                                    | stavalues2 | stavalues3 | stavalues4 | stavalues5 |
stadndistinct | staextinfo
----------+------------+-----------+------------+-------------+----------+-------------+----------+----------
+----------+----------+----------+--------+--------+--------+--------+--------+-------------+-------------
+-------------+------
-------+-------------
+-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------+-----------+-----------+-----------
+-----------+---------------+------------
   74069 | c         |         1 | f         |          0 |       4 |         -1 |       2 |       3 |       0 |       0 |
0 |     97 |     97 |      0 |      0 |      0 |           | {1}       |           |
      |           |
{1,130,260,390,520,650,780,910,1040,1170,1300,1430,1560,1690,1820,1950,2080,2210,2340,2470,2600,
2730,2860,2990,3120,3250,3380,3510,3640,3770,3900,4030,4160,4290,4420,4550,4680,4810,4940,5070
,5200,5330,5460,5590,57
20,5850,5980,6110,6240,6370,6500,6630,6760,6890,7020,7150,7280,7410,7540,7670,7800,7930,8060,8
190,8320,8450,8580,8710,8840,8970,9100,9230,9360,9490,9620,9750,9880,10010,10140,10270,10400,1
0530,10660,10790,10920,11050,11180,11310,11440,1
1570,11700,11830,11960,12090,12220,12350,12480,12610,12740,12870,13000} |           |
|           |           |            0 |
(1 row)
```

- pg_gtt_attached_pid(relOid)

  Description: Displays PIDs and session IDs of all threads that are using the specified global temporary table. If the thread pool is enabled and the session

that is using the global temporary table is in the detach state, the PID is **0** and **sessionid** indicates the session ID.

Parameter: OID of the global temporary table

Return type: record

Example:

```
gaussdb=# SELECT * FROM  pg_gtt_attached_pid(74069);
 relid |       pid       |   sessionid
-------+-----------------+-----------------
 74069 | 139648170456832 | 139648170456832
 74069 | 139648123270912 | 139648123270912
(2 rows)
```

- dbe_perf.get_global_full_sql_by_timestamp(start_timestamp timestamp, end_timestamp timestamp)

  Description: Obtains full SQL information at the instance level.

  Return type: record

  Table 7-149 dbe_perf.get_global_full_sql_by_timestamp parameter description

Parameter	Type	Description
start_timestamp	timestamp	Start point of the SQL start time range.
end_timestamp	timestamp	End point of the SQL start time range.

- dbe_perf.get_global_slow_sql_by_timestamp(start_timestamp timestamp, end_timestamp timestamp)

  Description: Obtains slow SQL information at the instance level.

  Return type: record

  Table 7-150 dbe_perf.get_global_slow_sql_by_timestamp parameter description

Parameter	Type	Description
start_timestamp	timestamp	Start point of the SQL start time range.
end_timestamp	timestamp	End point of the SQL start time range.

- statement_detail_decode(detail text, format text, pretty bool)

  Parses the details column in a full or slow SQL statement.

Table 7-151 statement_detail_decode parameter description

Parameter	Type	Description
detail	text	Set of events generated by the SQL statement (unreadable).
format	text	Parsing output format. The value is **plaintext** or **json**.
pretty	bool	Specifies whether to display the text in pretty format when **format** is set to **plaintext**. The options are as follows: • The value **true** indicates that events are separated by **\n**. • The value **false** indicates that events are separated by commas (,).

- pg_list_gtt_relfrozenxids()

  Description: Displays the frozen XID of each session.

  If the value of **pid** is **0**, the earliest frozen XID of all sessions is displayed.

  Parameter: none

  Return type: record

  Example:

```
gaussdb=# SELECT * FROM pg_list_gtt_relfrozenxids();
       pid       | relfrozenxid
-----------------+--------------
 139648123270912 |        11151
 139648170456832 |        11155
               0 |        11151
(3 rows)
```

7.6.33 Fault Injection System Function

- gs_fault_inject(int64, text, text, text, text, text)

  Description: This function cannot be called. WARNING information "unsupported fault injection" is reported when this function is called, which does not affect or change the database.

  Parameter: fault injection of the int64 type (**0**: CLOG extended page; **1**: CLOG page reading; **2**: forcible deadlock)

  – If the first input parameter of text is set to **2** and the second input parameter of text is set to **1**, the second input parameter deadlock occurs. Other input parameters are not deadlocked. When the first input parameter is **0** or **1**, the second input parameter indicates the number of the start page from which the CLOG starts to be extended or read.

  – The third input parameter of text indicates the number of extended or read pages when the first input parameter is **0** or **1**.

  – The fourth to sixth input parameters of text are reserved.

  Return type: int64

7.6.34 AI Feature Functions

- gs_index_advise(text)

  Description: Recommends an index for a single query statement.

  Parameter: SQL statement string

  Return type: record

  For details, see section "DBMind: Autonomous Database O&M > AI Sub-functions of DBMind > Index-advisor: Index Recommendation > Single-Query Index Recommendation" in *Feature Guide*.

- hypopg_create_index(text, [text])

  Description: Creates a virtual index.

  Parameter: (optional) character string of the statement for creating an index, level of the created virtual index

  Return type: record

  For details, see section "DBMind: Autonomous Database O&M > AI Sub-functions of DBMind > Index-advisor: Index Recommendation > Virtual Index" in *Feature Guide*.

- hypopg_display_index([text])

  Description: Displays information about all created virtual indexes.

  Parameter: (optional) level of the virtual index to be displayed

  Return type: record

  For details, see section "DBMind: Autonomous Database O&M > AI Sub-functions of DBMind > Index-advisor: Index Recommendation > Virtual Index" in *Feature Guide*.

- hypopg_drop_index(oid)

  Description: Deletes a specified virtual index.

  Parameter: OID of the index

  Return type: Boolean

  For details, see section "DBMind: Autonomous Database O&M > AI Sub-functions of DBMind > Index-advisor: Index Recommendation > Virtual Index" in *Feature Guide*.

- hypopg_reset_index([text])

  Description: Clears all virtual indexes.

  Parameter: (optional) level of the virtual index to be cleared

  Return type: none

  For details, see section "DBMind: Autonomous Database O&M > AI Sub-functions of DBMind > Index-advisor: Index Recommendation > Virtual Index" in *Feature Guide*.

- hypopg_estimate_size(oid)

  Description: Estimates the space required for creating a specified index.

  Parameter: OID of the index

  Return type: int8

For details, see section "DBMind: Autonomous Database O&M > AI Sub-functions of DBMind > Index-advisor: Index Recommendation > Virtual Index" in *Feature Guide*.

- check_engine_status(ip text, port text)

  Description: Tests whether a predictor engine provides services on a specified IP address and port number.

  Parameter: IP address and port number of the predictor engine.

  Return type: text

  NOTE

  This function is unavailable in the current version.

- encode_plan_node(optname text, orientation text, strategy text, options text, dop int8, quals text, projection text)

  Description: Encodes the plan operator information in the input parameters.

  Parameter: plan operator information

  Return type: text

  NOTE

  This function is an internal function. You are advised not to use it directly.

- model_train_opt(template text, model text)

  Description: Trains a given query performance prediction model.

  Parameters: template name and model name of the performance prediction model

  Return type: tartup_time_accuracy FLOAT8, total_time_accuracy FLOAT8, rows_accuracy FLOAT8, peak_memory_accuracy FLOAT8

  NOTE

  This function is unavailable in the current version.

- track_model_train_opt(ip text, port text)

  Description: Returns the training log address of the predictor engine with the specified IP address and port number.

  Parameter: IP address and port number of the predictor engine

  Return type: text

  NOTE

  This function is unavailable in the current version.

- encode_feature_perf_hist(datname text)

  Description: Encodes historical plan operators collected in the target database.

  Parameter: database name

  Return type: queryid bigint, plan_node_id int, parent_node_id int, left_child_id int, right_child_id int, encode text, startup_time bigint, total_time bigint, rows bigint, and peak_memory int

- gather_encoding_info(datname text)

  Description: Calls encode_feature_perf_hist to save the encoded data persistently.

Parameter: database name

Return type: int

- db4ai_predict_by_bool (text, VARIADIC "any")

  Description: Obtains a model whose return value is of the Boolean type for model inference. This function is an internal function. You are advised to use the PREDICT BY syntax for inference.

  Parameter: model name and input column name of the inference task

  Return type: Boolean

- db4ai_predict_by_float4(text, VARIADIC "any")

  Description: Obtains a model whose return value is of the float4 type for model inference. This function is an internal function. You are advised to use the PREDICT BY syntax for inference.

  Parameter: model name and input column name of the inference task

  Return type: float

- db4ai_predict_by_float8(text, VARIADIC "any")

  Description: Obtains a model whose return value is of the float8 type for model inference. This function is an internal function. You are advised to use the PREDICT BY syntax for inference.

  Parameter: model name and input column name of the inference task

  Return type: float

- db4ai_predict_by_int32(text, VARIADIC "any")

  Description: Obtains a model whose return value is of the int32 type for model inference. This function is an internal function. You are advised to use the PREDICT BY syntax for inference.

  Parameter: model name and input column name of the inference task

  Return type: int

- db4ai_predict_by_int64(text, VARIADIC "any")

  Description: Obtains a model whose return value is of the int64 type for model inference. This function is an internal function. You are advised to use the PREDICT BY syntax for inference.

  Parameter: model name and input column name of the inference task

  Return type: int

- db4ai_predict_by_numeric(text, VARIADIC "any")

  Description: Obtains a model whose return value is of the numeric type for model inference. This function is an internal function. You are advised to use the PREDICT BY syntax for inference.

  Parameter: model name and input column name of the inference task

  Return type: numeric

- db4ai_predict_by_text(text, VARIADIC "any")

  Description: Obtains a model whose return value is of the character type for model inference. This function is an internal function. You are advised to use the PREDICT BY syntax for inference.

  Parameter: model name and input column name of the inference task

  Return type: text

- db4ai_predict_by_float8_array(text, VARIADIC "any")

  Description: Obtains a model whose return value is of the character type for model inference. This function is an internal function. You are advised to use the PREDICT BY syntax for inference.

  Parameter: model name and input column name of the inference task

  Return type: text

- gs_explain_model(text)

  Description: Obtains the model whose return value is of the character type for text-based model parsing.

  Parameter: model name

  Return type: text

- gs_ai_stats_explain(text, text[])

  Description: Prints the intelligent statistics of multiple columns in the corresponding table and columns.

  Parameters: table name and column name collection.

  Return type: text

- gs_acm_analyze_workload_manual()

  Description: Manually trains the cardinality estimation model based on the operator feedback data in the current database.

  Parameter: none

  Return type: text

- gs_stat_get_acm_feedback_operator_info()

  Description: Displays all operator feedback data collected in the global memory.

  Parameter: none

  Return type: record

- gs_stat_get_sql_feedback_info()

  Description: Displays the number of times that SQL statements are used and the execution time when SQL statements are used and not used within the effective range of the feedback cardinality function in the global memory.

  Parameter: none

  Return type: record

- gs_costmodel_calibration_manual()

  Description: Manually triggers operator time collection and corrects cost model parameters.

  Parameter: none

  Return type: text

- gs_show_aplan(cstring)

  Description: Views the multi-plan cache of queries cached in the current session.

  Parameter: prepare name is used to search for the corresponding cache plan.

  Return type: text

- ai_watchdog_detection_warnings()

Description: Obtains the risk alarm information of the AI watchdog. The SYSADMIN or MONADMIN access permission is required.

Parameter: none

Return type: record

- ai_watchdog_monitor_status(int)

  Description: Obtains the monitoring information of the AI watchdog. The SYSADMIN or MONADMIN access permission is required.

  Parameter: Returns the upper limit of the length of the monitored sequence.

  Return type: record

- ai_watchdog_parameters()

  Description: Obtains the internal parameters or status information of the AI watchdog. The SYSADMIN or MONADMIN access permission is required.

  Parameter: none

  Return type: record

7.6.35 Sensitive Data Discovery Function

- gs_sensitive_data_discovery(scan_target text, scan_classifier text)

  Description: Scans target data and returns statistical scanning results.

  Parameters:

  scan_target: specifies the object to be scanned. The value must be the name of a schema, table, or column, and the upper-level names of the object to be scanned must be specified. For example, if a column is scanned, the object to be scanned is *scheme_name*.*table_name*.*column_name*.

  scan_classifier: specifies the classifier to be used. Five classifiers can be specified: email, creditcard, phonenumber, chinesename, and encryptedcontent. Multiple classifiers are separated by commas (,). The value **all** indicates all classifiers are selected.

  Return type: record

  For details, see section "Sensitive Data Discovery" in *Feature Guide*.

- gs_sensitive_data_discovery_detail(scan_target text, scan_classifier text)

  Description: Scans the target data and returns the detailed scanning result.

  scan_target: specifies the object to be scanned. The value must be the name of a schema, table, or column, and the upper-level names of the object to be scanned must be specified. For example, if a column is scanned, the object to be scanned is *scheme_name*.*table_name*.*column_name*.

  scan_classifier: specifies the classifier to be used. Five classifiers can be specified: email, creditcard, phonenumber, chinesename, and encryptedcontent. Multiple classifiers are separated by commas (,). The value **all** indicates all classifiers are selected.

  Return type: record

  For details, see section "Sensitive Data Discovery" in *Feature Guide*.

7.6.36 Dynamic Data Masking Functions

NOTE

This function is an internal function.

- creditcardmasking(col text, letter char default 'x')

  Description: Replaces the digits before the last four bits following the col string with letters.

  Parameter: Character string to be replaced or character string used for replacement

  Return type: text

- basicmailmasking(col text, letter char default 'x')

  Description: Replaces the characters before the first at sign (@) in the col string with letters.

  Parameter: Character string to be replaced or character string used for replacement

  Return type: text

- fullmailmasking(col text, letter char default 'x')

  Description: Replaces the characters (except **@**) before the last period (.) in the col string with letters.

  Parameter: Character string to be replaced or character string used for replacement

  Return type: text

- alldigitsmasking(col text, letter char default '0')

  Description: Replaces the digits in the col string with letters.

  Parameter: Character string to be replaced or character string used for replacement

  Return type: text

- shufflemasking(col text)

  Description: Sorts the characters in the col string out of order.

  Parameter: Character string to be replaced or character string used for replacement

  Return type: text

- randommasking(col text)

  Description: Randomizes the characters in the col string.

  Parameter: Character string to be replaced or character string used for replacement

  Return type: text

- regexpmasking

  Description: Specifies the internal function of the masking policy, which is used to replace characters using a regular expression.

  Parameter: col text, reg text, replace_text text, pos INTEGER default 0, reg_len INTEGER default -1

  Return type: text

7.6.37 Hierarchical Recursion Query Functions

The following functions can be used in a hierarchical recursion query statement to return information about the connection path.

- sys_connect_by_path(col, separator)

  Description: Returns the connection path from the root node to the current row. This function applies only to hierarchical recursion queries.

  The **col** parameter indicates the name of the column displayed in the path. Only columns of the CHAR, VARCHAR, NVARCHAR2, TEXT, INT1, INT2, INT4, INT8, FLOAT4, FLOAT8, or NUMERIC type are supported. The **separator** parameter indicates the separator between path nodes. Currently, the col input of the function does not support expression input.

  Return type: text

  Example:

```
-- Create a table and insert data.
gaussdb=# CREATE TABLE connect_by_table(id int,pid int,name text);
CREATE TABLE
gaussdb=# INSERT INTO connect_by_table VALUES(1,0,'a'),(2,1,'b'),(3,2,'c'),(4,1,'d');
INSERT 0 4
-- Query.
gaussdb=# SELECT *, sys_connect_by_path(name, '-') FROM connect_by_table START WITH id = 1
CONNECT BY prior id = pid ORDER BY pid, id;
 id | pid | name | sys_connect_by_path
----+-----+------+---------------------
  1 |   0 | a    | -a
  2 |   1 | b    | -a-b
  4 |   1 | d    | -a-d
  3 |   2 | c    | -a-b-c
(4 rows)
-- Restore the environment.
gaussdb=# DROP TABLE IF EXISTS connect_by_table;
DROP TABLE
```

- connect_by_root(col)

  Description: Returns the value of a column in the top-most parent row of the current row. This function applies only to hierarchical recursion queries.

  The **col** parameter indicates the name of an output column.

  Return type: data type of the specified column **col**.

  Example:

```
-- Create a table and insert data.
gaussdb=# CREATE TABLE connect_by_table(id int,pid int,name text);
CREATE TABLE
gaussdb=# INSERT INTO connect_by_table VALUES(1,0,'a'),(2,1,'b'),(3,2,'c'),(4,1,'d');
INSERT 0 4
-- Query.
gaussdb=# SELECT *, connect_by_root(name) FROM connect_by_table START WITH id = 1 CONNECT
BY prior id = pid ORDER BY pid, id;
 id | pid | name | connect_by_root
----+-----+------+-----------------
  1 |   0 | a    | a
  2 |   1 | b    | a
  4 |   1 | d    | a
  3 |   2 | c    | a
(4 rows)
-- Restore the environment.
gaussdb=# drop table if exists connect_by_table;
DROP TABLE
```


7.6.38 Other System Functions

Built-in functions and operators of GaussDB are compatible with PostgreSQL.

_pg_char_max_length	_pg_char_octet_length	_pg_datetime_precision	_pg_expandarray	_pg_index_position	_pg_interval_type	_pg_numeric_precision
_pg_numeric_precision_radix	_pg_numeric_scale	_pg_truetypid	_pg_truetypmod	abbrev	abs	abstime
abstimeeq	abstimege	abstimegt	abstimein	abstimele	abstimelt	abstimene
abstimeout	abstimerecv	abstimesend	aclcontains	acldefault	aclexplode	aclinsert
aclitemeq	aclitemin	aclitemout	aclremove	acos	age	akeys
any_in	any_out	anyarray_in	anyarray_out	anyarray_recv	anyarray_send	anyelement_in
anyelement_out	anyenum_in	anyenum_out	anynonarray_in	anynonarray_out	anyrange_in	anyrange_out
anytextcat	area	areajoinsel	areasel	array_agg	array_agg_finalfn	array_agg_transfn
array_append	array_cat	array_dims	array_eq	array_fill	array_ge	array_gt
array_in	array_larger	array_le	array_length	array_lower	array_lt	array_ndims
array_ne	array_out	array_prepend	array_recv	array_send	array_smaller	array_to_json
array_to_string	array_typanalyze	array_upper	arraycontained	arraycontains	arraycontjoinsel	arraycontsel
arrayoverlap	ascii	asin	atan	atan2	avals	avg
big5_to_euc_tw	big5_to_mic	big5_to_utf8	bit	bit_and	bit_in	bit_length
bit_or	bit_out	bit_recv	bit_send	bitand	bitcat	bitcmp
biteq	bitge	bitgt	bitle	bitlt	bitne	bitnot

bitor	bitshiftleft	bitshiftright	bittypmodin	bittypmodout	bitxor	bool
bool_and	bool_or	booland_statefunc	booleq	boolge	boolgt	boolin
boolle	boollt	boolne	boolor_statefunc	boolout	boolrecv	boolsend
box	box_above	box_above_eq	box_add	box_below	box_below_eq	box_center
box_contain	box_contain_pt	box_contained	box_distance	box_div	box_eq	box_ge
box_gt	box_in	box_intersect	box_le	box_left	box_lt	box_mul
box_out	box_overabove	box_overbelow	box_overlap	box_overleft	box_overright	box_recv
box_right	box_same	box_send	box_sub	bpchar	bpchar_larger	bpchar_pattern_ge
bpchar_pattern_gt	bpchar_pattern_le	bpchar_pattern_lt	bpchar_smaller	bpchar_sortsupport	bpcharcmp	bpchareq
bpcharge	bpchargt	bpcharicliike	bpcharicnlike	bpcharicregexeq	bpcharicregexne	bpcharin
bpcharle	bpcharlike	bpcharlt	bpcharne	bpcharnlike	bpcharout	bpcharrecv
bpcharregexeq	bpcharregexne	bpcharsend	bpchartypmodin	bpchartypmodout	broadcast	btabstimecmp
btarraycmp	btbeginscan	btboolcmp	btbpchar_pattern_cmp	btbuild	btbuildempty	btbulkdelete
btcanreturn	btcharcmp	btcostestimate	btendscan	btfloat48cmp	btfloat4cmp	btfloat4sortsupport
btfloat84cmp	btfloat8cmp	btfloat8sortsupport	btgetbitmap	btgettuple	btinsert	btint24cmp
btint28cmp	btint2cmp	btint2sortsupport	btint42cmp	btint48cmp	btint4cmp	btint4sortsupport

btint82cmp	btint84cmp	btint8cmp	btint8sortsupport	btmarkpos	btnamecmp	btnamesortsupport
btoidcmp	btoidsortsupport	btoidvectorcmp	btoptions	btrecordcmp	btreltimecmp	btrescan
btrestrpos	btrim	bttext_pattern_cmp	bttextcmp	bttextsortsupport	bttidcmp	bttintervalcmp
btvacuumcleanup	bytea_sortsupport	bytea_string_agg_finalfn	bytea_string_agg_transfn	byteacat	byteacmp	byteaeq
byteage	byteagt	byteain	byteale	bytealike	bytealt	byteane
byteanlike	byteaout	bytearecv	byteasend	cash_cmp	cash_div_cash	cash_div_flt4
cash_div_flt8	cash_div_int2	cash_div_int4	cash_div_int8	cash_eq	cash_ge	cash_gt
cash_in	cash_le	cash_lt	cash_mi	cash_mul_flt4	cash_mul_flt8	cash_mul_int2
cash_mul_int4	cash_mul_int8	cash_ne	cash_out	cash_pl	cash_recv	cash_send
cashlarger	cashsmaller	cbrt	ceil	ceiling	center	char
char_length	character_length	chareq	charge	chargt	charin	charle
charlt	charne	charout	charrecv	charsend	chr	cideq
cidin	cidout	cidr	cidr_in	cidr_out	cidr_recv	cidr_send
cidrecv	cidsend	circle	circle_above	circle_add_pt	circle_below	circle_center
circle_contain	circle_contain_pt	circle_contained	circle_distance	circle_div_pt	circle_eq	circle_ge
circle_gt	circle_in	circle_le	circle_left	circle_lt	circle_mul_pt	circle_ne

circle_out	circle_overabove	circle_overbelow	circle_overlap	circle_overleft	circle_overright	circle_recv
circle_right	circle_same	circle_send	circle_sub_pt	clock_timestamp	close_lb	close_ls
close_lseg	close_pb	close_pl	close_ps	close_sb	close_sl	col_description
concat	concat_ws	contjoinsel	contsel	convert	convert_from	convert_to
corr	cos	cot	count	covar_pop	covar_samp	cstring_in
cstring_out	cstring_recv	cstring_send	cume_dist	current_database	current_query	current_schema
-	current_setting	current_user	currtid	currtid2	currval	-
-	-	-	-	date	date_cmp	date_cmp_timestamp
date_cmp_timestamptz	date_eq	date_eq_timestamp	date_eq_timestamptz	date_ge	date_ge_timestamp	date_ge_timestamptz
date_gt	date_gt_timestamp	date_gt_timestamptz	date_in	date_larger	date_le	date_le_timestamp
date_le_timestamptz	date_lt	date_lt_timestamp	date_lt_timestamptz	date_mi	date_mi_interval	date_mii
date_ne	date_ne_timestamp	date_ne_timestamptz	date_out	date_pl_interval	date_pli	date_recv
date_send	date_smaller	date_sortsupport	daterange_canonical	daterange_subdiff	datetime_pl	datetimetz_pl
dcbrt	decode	defined	degrees	delete	dense_rank	dexp
diagonal	diameter	dispell_init	dispell_lexize	dist_cpoly	dist_lb	dist_pb
dist_pc	dist_pl	dist_ppath	dist_ps	dist_sb	dist_sl	div

dlog1	dlog10	domain_in	domain_recv	dpow	dround	dsimple_init
dsimple_lexize	dsnowball_init	dsnowball_lexize	dsqrt	dsynonym_init	dsynonym_lexize	dtrunc
each	enum_ne	enum_out	enum_range	enum_recv	enum_send	enum_smaller
eqjoinsel	eqsel	euc_cn_to_mic	euc_cn_to_utf8	euc_jis_2004_to_shift_jis_2004	euc_jis_2004_to_utf8	euc_jp_to_mic
euc_jp_to_sjis	euc_jp_to_utf8	euc_kr_to_mic	euc_kr_to_utf8	euc_tw_to_big5	euc_tw_to_mic	euc_tw_to_utf8
every	exist	exists_all	exists_any	exp	factorial	family
fdw_handler_in	fdw_handler_out	fetchval	first_value	float4	float4_accum	float48div
float48eq	float48ge	float48gt	float48le	float48lt	float48mi	float48mul
float48ne	float48pl	float4abs	float4div	float4eq	float4ge	float4gt
float4in	float4larger	float4le	float4lt	float4mi	float4mul	float4ne
float4out	float4pl	float4recv	float4send	float4smaller	float4um	float4up
float8	float8_accum	float8_avg	float8_collect	float8_corr	float8_covar_pop	float8_covar_samp
float8_regr_accum	float8_regr_avgx	float8_regr_avgy	float8_regr_collect	float8_regr_intercept	float8_regr_r2	float8_regr_slope
float8_regr_sxx	float8_regr_sxy	float8_regr_syy	float8_stddev_pop	float8_stddev_samp	float8_var_pop	float8_var_samp
float84div	float84eq	float84ge	float84gt	float84le	float84lt	float84mi
float84mul	float84ne	float84pl	float8abs	float8div	float8eq	float8ge
float8gt	float8in	float8larger	float8le	float8lt	float8mi	float8mul

float8ne	float8out	float8pl	float8recv	float8send	float8smaller	float8um
float8up	floor	flt4_mul_cash	flt8_mul_cash	fmgr_c_validator	fmgr_internal_validator	fmgr_sql_validator
format	format_type	gb18030_to_utf8	gbk_to_utf8	generate_series	generate_subscripts	get_bit
get_byte	get_current_ts_config	-	-	-	-	-
gtsquery_compress	gtsquery_consistent	gtsquery_decompress	gtsquery_penalty	gtsquery_picksplit	gtsquery_same	gtsquery_union
gtsvector_compress	gtsvector_consistent	gtsvector_decompress	gtsvector_penalty	gtsvector_picksplit	gtsvector_same	gtsvector_union
gtsvectorin	gtsvectorout	has_tablespace_privilege	has_type_privilege	hash_aclitem	hashbeginscan	hashbuild
hashbuildempty	hashbulkdelete	hashcostestimate	hashendscan	hashgetbitmap	hashgettuple	hashinsert
hashint2vector	hashint4	hashint8	hashmacaddr	hashmarkpos	hashname	hashoid
hashoidvector	hashoptions	hashrescan	hashrestrpos	hashtext	hashvacuumcleanup	hashvarlena
host	hostmask	iclikejoinsel	iclikesel	icnlikejoinsel	icnlikesel	icregexeqjoinsel
icregexeqsel	icregexnejoinsel	icregexnesel	inet_client_addr	inet_client_port	inet_in	inet_out
inet_recv	inet_send	inet_server_addr	inet_server_port	inetand	inetmi	inetmi_int8
inetnot	inetor	inetpl	initcap	int2_accum	int2_avg_accum	int2_mul_cash
int2_sum	int24div	int24eq	int24ge	int24gt	int24le	int24lt
int24mi	int24mul	int24ne	int24pl	int28div	int28eq	int28ge

int28gt	int28le	int28lt	int28mi	int28mul	int28ne	int28pl
int2abs	int2and	int2div	int2eq	int2ge	int2gt	int2in
int2larger	int2le	int2lt	int2mi	int2mod	int2mul	int2ne
int2not	int2or	int2out	int2pl	int2recv	int2send	int2shl
int2shr	int2smaller	int2um	int2up	int2vectoreq	int2vectorin	int2vectorout
int2vectorrecv	int2vectorsend	int2xor	int4_accum	int4_avg_accum	int4_mul_cash	int4_sum
int42div	int42eq	int42ge	int42gt	int42le	int42lt	int42mi
int42mul	int42ne	int42pl	int48div	int48eq	int48ge	int48gt
int48le	int48lt	int48mi	int48mul	int48ne	int48pl	int4abs
int4and	int4div	int4eq	int4ge	int4gt	int4in	int4inc
int4larger	int4le	int4lt	int4mi	int4mod	int4mul	int4ne
int4not	int4or	int4out	int4pl	int4range	int4range_canonical	int4range_subdiff
int4recv	int4send	int4shl	int4shr	int4smaller	int4um	int4up
int4xor	int8	int8_avg	int8_avg_accum	int8_avg_collect	int8_mul_cash	int8_sum
int8_sum_to_int8	int8_accum	int82div	int82eq	int82ge	int82gt	int82le
int82lt	int82mi	int82mul	int82ne	int82pl	int84div	int84eq
int84ge	int84gt	int84le	int84lt	int84mi	int84mul	int84ne
int84pl	int8abs	int8and	int8div	int8eq	int8ge	int8gt
int8in	int8inc	int8inc_any	int8inc_float8_float8	int8larger	int8le	int8lt

int8mi	int8mod	int8mul	int8ne	int8not	int8or	int8out
int8pl	int8pl_inet	int8range	int8range_canonical	int8range_subdiff	int8recv	int8send
int8shl	int8shr	int8smaller	int8um	int8up	int8xor	integer_pl_date
inter_lb	inter_sb	inter_sl	internal_in	internal_out	interval	interval_accum
interval_avg	interval_cmp	interval_collect	interval_div	interval_eq	interval_ge	interval_gt
interval_hash	interval_in	interval_larger	interval_le	interval_lt	interval_mi	interval_mul
interval_ne	interval_out	interval_pl	interval_pl_date	interval_pl_time	interval_pl_timestamp	interval_pl_timestamptz
interval_pl_timetz	interval_recv	interval_send	interval_smaller	interval_transform	interval_um	intervaltypmodin
intervaltypmodout	intinterval	isexists	ishorizontal	iso_to_koi8r	iso_to_mic	iso_to_win1251
iso_to_win866	iso8859_1_to_utf8	iso8859_to_utf8	isparallel	isperp	isvertical	johab_to_utf8
jsonb_in	jsonb_out	jsonb_recv	jsonb_send	-	-	-
json_in	json_out	json_recv	json_send	justify_days	justify_hours	justify_interval
koi8r_to_iso	koi8r_to_mic	koi8r_to_utf8	koi8r_to_win1251	koi8r_to_win866	koi8u_to_utf8	language_handler_in
language_handler_out	latin1_to_mic	latin2_to_mic	latin2_to_win1250	latin3_to_mic	latin4_to_mic	like_escape
likejoinsel	likesel	line	line_distance	line_eq	line_horizontal	line_in
line_interpt	line_intersect	line_out	line_parallel	line_perp	line_recv	line_send
line_vertical	ln	lo_close	lo_creat	lo_create	lo_export	lo_import

lo_lseek	lo_open	lo_tell	lo_truncate	lo_unlink	log	loread
lower	lower_inc	lower_inf	lowrite	lpad	lseg	lseg_center
lseg_distance	lseg_eq	lseg_ge	lseg_gt	lseg_horizontal	lseg_in	lseg_interpt
lseg_intersect	lseg_le	lseg_length	lseg_lt	lseg_ne	lseg_out	lseg_parallel
lseg_perp	lseg_recv	lseg_send	lseg_vertical	ltrim	macaddr_and	macaddr_cmp
macaddr_eq	macaddr_ge	macaddr_gt	macaddr_in	macaddr_le	macaddr_lt	macaddr_ne
macaddr_not	macaddr_or	macaddr_out	macaddr_recv	macaddr_send	makeaclitem	masklen
max	md5 (The MD5 encryption algorithm has lower security and poses security risks. Therefore, you are advised to use a more secure encryption algorithm.)	mic_to_big5	mic_to_euc_cn	mic_to_euc_jp	mic_to_euc_kr	mic_to_euc_tw
mic_to_iso	mic_to_koi8r	mic_to_latin1	mic_to_latin2	mic_to_latin3	mic_to_latin4	mic_to_sjis
mic_to_win1250	mic_to_win1251	mic_to_win866	min	mktinterval	money	mul_d_interval
name	nameeq	namege	namegt	nameiclike	nameicnlike	nameicregexeq
nameicregexne	namein	namele	namelike	namelt	namene	namenlike

nameout	namerecv	nameregexeq	nameregexne	namesend	neqjoinsel	neqsel
network_cmp	network_eq	network_ge	network_gt	network_le	network_lt	network_ne
network_sub	network_subeq	network_sup	network_supeq	nlikejoinsel	nlikesel	numeric
numeric_abs	numeric_accum	numeric_add	numeric_avg	numeric_avg_accum	numeric_avg_collect	numeric_cmp
numeric_collect	numeric_div	numeric_div_trunc	numeric_eq	numeric_exp	numeric_fac	numeric_ge
numeric_gt	numeric_in	numeric_inc	numeric_larger	numeric_le	numeric_ln	numeric_log
numeric_lt	numeric_mod	numeric_mul	numeric_ne	numeric_out	numeric_power	numeric_recv
numeric_send	numeric_smaller	numeric_sortsupport	numeric_sqrt	numeric_stddev_pop	numeric_stddev_samp	numeric_sub
numeric_transform	numeric_uminus	numeric_uplus	numeric_var_pop	numeric_var_samp	numerictypmodin	numerictypmodout
numrange_subdiff	oid	oideq	oidge	oidgt	oidin	oidlarger
oidle	oidlt	oidne	oidout	oidrecv	oidsend	oidsmaller
oidvectoreq	oidvectorge	oidvectorgt	oidvectorin	oidvectorle	oidvectorlt	oidvectorne
oidvectorout	oidvectorrecv	oidvectorsend	oidvectortypes	on_pb	on_pl	on_ppath
on_ps	on_sb	on_sl	opaque_in	opaque_out	ordered_set_transition	overlaps
overlay	path	path_add	path_add_pt	path_center	path_contain_pt	path_distance

path_div_pt	path_in	path_inter	path_length	path_mul_pt	path_n_eq	path_n_ge
path_n_gt	path_n_le	path_n_lt	path_npoints	path_out	path_recv	path_send
path_sub_pt	percentile_cont	percentile_cont_float8_final	percentile_cont_interval_final	pg_char_to_encoding	pg_cursor	pg_encoding_max_length
pg_encoding_to_char	pg_extension_config_dump	-	-	pg_node_tree_in	pg_node_tree_out	pg_node_tree_recv
pg_node_tree_send	pg_prepared_statement	pg_prepared_xact	-	-	pg_show_all_settings	pg_stat_get_bgwriter_stat_reset_time
pg_stat_get_buf_fsync_backend	pg_stat_get_checkpoint_sync_time	pg_stat_get_checkpoint_write_time	pg_stat_get_db_blk_read_time	pg_stat_get_db_blk_write_time	pg_stat_get_db_conflict_all	pg_stat_get_db_conflict_bufferpin
pg_stat_get_db_conflict_snapshot	pg_stat_get_db_conflict_startup_deadlock	pg_switch_xlog	-	pg_timezone_abbrevs	pg_timezone_names	pg_stat_get_wal_receiver
plpgsql_call_handler	plpgsql_inline_handler	plpgsql_validator	point_above	point_add	point_below	point_distance
point_div	point_eq	point_horiz	point_in	point_left	point_mul	point_ne
point_out	point_recv	point_right	point_send	point_sub	point_vert	poly_above
poly_below	poly_center	poly_contain	poly_contain_pt	poly_contained	poly_distance	poly_in
poly_left	poly_npoints	poly_out	poly_overabove	poly_overbelow	poly_overlap	poly_overleft
poly_overright	poly_recv	poly_right	poly_same	poly_send	polygon	position
positionjoinsel	positionsel	postgresql_fdw_validator	pow	power	prsd_end	prsd_headline

prsd_lextype	prsd_nexttoken	prsd_start	pt_contained_circle	pt_contained_poly	-	-
-	quote_ident	quote_literal	quote_nullable	radians	radius	random
range_adjacent	range_after	range_before	range_cmp	range_contained_by	range_contains	range_contains_elem
range_ge	range_eq	range_gt	range_in	range_intersect	range_le	range_lt
range_minus	range_ne	range_out	range_overlaps	range_overleft	range_overright	range_recv
range_send	range_typanalyze	range_union	rank	record_eq	record_ge	record_gt
record_in	record_le	record_lt	record_ne	record_out	record_recv	record_send
regclass	regclassin	regclassout	regclassrecv	regclasssend	regconfigin	regconfigout
regconfigrecv	regconfigsend	regdictionaryin	regdictionaryout	regdictionaryrecv	regdictionarysend	regexeqjoinsel
regexeqsel	regexnejoinsel	regexnesel	regexp_matches	regexp_replace	regexp_split_to_array	regexp_split_to_table
regoperatorin	regoperatorout	regoperatorrecv	regoperatorsend	regoperin	regoperout	regoperrecv
regopersend	regprocedurein	regprocedureout	regprocedurerecv	regproceduresend	regprocin	regprocout
regprocrecv	regprocsend	regr_avgx	regr_avgy	regr_count	regr_intercept	regr_r2
regr_slope	regr_sxx	regr_sxy	regr_syy	regtypein	regtypeout	regtyperecv
regtypesend	reltime	reltimeeq	reltimege	reltimegt	reltimein	reltimele
reltimelt	reltimene	reltimeout	reltimerecv	reltimesend	repeat	replace

reverse	RI_FKey_cascade_del	RI_FKey_cascade_upd	RI_FKey_check_ins	RI_FKey_check_upd	RI_FKey_noaction_del	RI_FKey_noaction_upd
RI_FKey_restrict_del	RI_FKey_restrict_upd	RI_FKey_setdefault_del	RI_FKey_setdefault_upd	RI_FKey_setnull_del	RI_FKey_setnull_upd	right
round	row_number	row_to_json	rpad	rtrim	scalargtjoinsel	scalargtsel
scalarltjoinsel	scalarltsel	-	schema_to_xml_and_xmlschema	-	session_user	set_bit
set_byte	set_config	set_masklen	shift_jis_2004_to_euc_jis_2004	shift_jis_2004_to_utf8	sjis_to_euc_jp	sjis_to_mic
sjis_to_utf8	smgrin	smgrout	spg_kd_choose	spg_kd_config	spg_kd_inner_consistent	spg_kd_picksplit
spg_quad_choose	spg_quad_config	spg_quad_inner_consistent	spg_quad_leaf_consistent	spg_quad_picksplit	spg_text_choose	spg_text_config
spg_text_inner_consistent	spg_text_leaf_consistent	spg_text_picksplit	spgbeginscan	spgbuild	spgbuildempty	spgbulkdelete
spgcanreturn	spgcostestimate	spgendscan	spggetbitmap	spggettuple	spginsert	spgmarkpos
spgoptions	spgrescan	spgrestrpos	spgvacuumcleanup	stddev	stddev_pop	stddev_samp
string_agg	string_agg_finalfn	string_agg_transfn	strip	sum	suppress_redundant_updates_trigger	-
-	-	tan	text	text_ge	text_gt	text_larger

text_le	text_lt	text_pattern_ge	text_pattern_gt	text_pattern_le	text_pattern_lt	text_smaller
textanycat	textcat	texteq	texticlike	texticnlike	texticregexeq	texticregexne
textin	textlike	textne	textnlike	textout	textrecv	textregexeq
textregexne	textsend	thesaurus_init	thesaurus_lexize	tideq	tidge	tidgt
tidin	tidlarger	tidle	tidlt	tidne	tidout	tidrecv
tidsend	tidsmaller	time	time_cmp	time_eq	time_ge	time_gt
time_hash	time_in	time_larger	time_le	time_lt	time_mi_interval	time_mi_time
time_ne	time_out	time_pl_interval	time_recv	time_send	time_smaller	time_transform
timedate_pl	timemi	timepl	timestamp	timestamp_cmp	timestamp_cmp_date	timestamp_cmp_timestamptz
timestamp_eq	timestamp_eq_date	timestamp_eq_timestamptz	timestamp_ge	timestamp_ge_date	timestamp_ge_timestamptz	timestamp_gt
timestamp_gt_date	timestamp_gt_timestamptz	timestamp_hash	timestamp_in	timestamp_larger	timestamp_le	timestamp_le_date
timestamp_le_timestamptz	timestamp_lt	timestamp_lt_date	timestamp_lt_timestamptz	timestamp_mi	timestamp_mi_interval	timestamp_ne
timestamp_ne_date	timestamp_ne_timestamptz	timestamp_out	timestamp_pl_interval	timestamp_recv	timestamp_send	timestamp_smaller
timestamp_sortsupport	timestamp_transform	timestamptypmodin	timestamptypmodout	timestamptz	timestamptz_cmp	timestamptz_cmp_date

timestamptz_cmp_timestamp	timestamptz_eq	timestamptz_eq_date	timestamptz_eq_timestamp	timestamptz_ge	timestamptz_ge_date	timestamptz_ge_timestamp
timestamptz_gt	timestamptz_gt_date	timestamptz_gt_timestamp	timestamptz_in	timestamptz_larger	timestamptz_le	timestamptz_le_date
timestamptz_le_timestamp	timestamptz_lt	timestamptz_lt_date	timestamptz_lt_timestamp	timestamptz_mi	timestamptz_mi_interval	timestamptz_ne
timestamptz_ne_date	timestamptz_ne_timestamp	timestamptz_out	timestamptz_pl_interval	timestamptz_recv	timestamptz_send	timestamptz_smaller
timestamptztypmodin	timestamptztypmodout	timetypmodin	timetypmodout	timetz	timetz_cmp	timetz_eq
timetz_ge	timetz_gt	timetz_hash	timetz_in	timetz_larger	timetz_le	timetz_lt
timetz_mi_interval	timetz_ne	timetz_out	timetz_pl_interval	timetz_recv	timetz_send	timetz_smaller
timetzdate_pl	timetztypmodin	timetztypmodout	timezone (2069)	timezone (1159)	timezone (2037)	timezone (2070)
timezone (1026)	timezone (2038)	tintervalct	tintervaleq	tintervalge	tintervalgt	tintervalin
tintervalle	tintervalleneq	tintervallenge	tintervallengt	tintervallenle	tintervallenlt	tintervallenne
tintervallt	tintervalne	tintervalout	tintervalov	tintervalrecv	tintervalsame	tintervalsend
tintervalstart	to_ascii (1845)	to_ascii (1847)	to_ascii (1846)	trigger_in	trigger_out	ts_match_qv
ts_match_tq	ts_match_tt	ts_match_vq	ts_rank	ts_rank_cd	ts_rewrite	ts_stat
ts_token_type	ts_typanalyze	tsmatchjoinsel	tsmatchsel	tsq_mcontained	tsq_mcontains	tsquery_and
tsquery_cmp	tsquery_eq	tsquery_ge	tsquery_gt	tsquery_le	tsquery_lt	tsquery_ne

tsquery_not	tsquery_or	tsqueryin	tsqueryout	tsqueryrecv	tsquerysend	tsrange
tsrange_subdiff	tstzrange	tstzrange_subdiff	tsvector_cmp	tsvector_concat	tsvector_eq	tsvector_ge
tsvector_gt	tsvector_le	tsvector_lt	tsvector_ne	tsvector_update_trigger	tsvector_update_trigger_column	tsvectorin
tsvectorout	tsvectorrecv	tsvectorsend	txid_current	txid_current_snapshot	txid_snapshot_in	txid_snapshot_out
txid_snapshot_recv	txid_snapshot_send	txid_snapshot_xip	txid_snapshot_xmax	txid_snapshot_xmin	txid_visible_in_snapshot	uhc_to_utf8
unique_key_recheck	unknownin	unknownout	unknownrecv	unknownsend	unnest	utf8_to_big5
utf8_to_euc_cn	utf8_to_euc_jis_2004	utf8_to_euc_jp	utf8_to_euc_kr	utf8_to_euc_tw	utf8_to_gb18030	utf8_to_gbk
utf8_to_iso8859	utf8_to_iso8859_1	utf8_to_johab	utf8_to_koi8r	utf8_to_koi8u	utf8_to_shift_jis_2004	utf8_to_sjis
utf8_to_uhc	utf8_to_win	uuid_cmp	uuid_eq	uuid_ge	uuid_gt	uuid_hash
uuid_in	uuid_le	uuid_lt	uuid_ne	uuid_out	uuid_recv	uuid_send
var_pop	var_samp	varbit	varbit_in	varbit_out	varbit_recv	varbit_send
varbit_transform	varbitcmp	varbiteq	varbitge	varbitgt	varbitle	varbitlt
varbitne	varbittypmodin	varbittypmodout	varchar	varchar_transform	varcharin	varcharout
varcharrecv	varcharsend	varchartypmodin	varchartypmodout	variance	void_in	void_out
void_recv	void_send	win_to_utf8	win1250_to_latin2	win1250_to_mic	win1251_to_iso	win1251_to_koi8r

win1251_to_mic	win1251_to_win866	win866_to_iso	win866_to_koi8r	win866_to_mic	win866_to_win1251	xideq
xideqint4	xidin	xidout	xidrecv	xidsend	xml	xml_in
-	-	-	xml_out	xml_recv	xml_send	-
-	xmlconcat2	-	xmlvalidate	pg_notify	year_in	year_out
year_recv	year_send	yeartypmodin	yeartypmodout	year_eq	year_ne	year_lt
year_le	year_gt	year_ge	year_cmp	year_hash	year_larger	year_smaller
year_mi	year_mi_int4	int4_mi_year	year_pl	year_pl_int4	int4_pl_year	int4_year
year_int4	date_year	numeric_year	text_year	time_year	timestamp_year	timestamptz_year
lpad_s	int8numericle	int8numericge	btint12cmp	btint14cmp	btint18cmp	btint116cmp
btint1numericcmp	btint21cmp	btint216cmp	btint2numericcmp	btint41cmp	btint416cmp	btint4numericcmp
btint81cmp	btint816cmp	btint8numericcmp	btint161cmp	btint162cmp	btint164cmp	btint168cmp
btnumericint1cmp	btnumericint2cmp	btnumericint4cmp	btnumericint8cmp	btint16cmp	hashint16	hashint1_numeric
hashint2_numeric	hashint4_numeric	hashint8_numeric	int12eq	int12ne	int12lt	int12gt
int12le	int12ge	int14eq	int14ne	int14lt	int14gt	int14le
int14ge	int18eq	int18ne	int18lt	int18gt	int18le	int18ge
int116eq	int116ne	int116lt	int116gt	int116le	int116ge	int1numericeq
int1numericne	int1numericlt	int1numericgt	int1numericle	int1numericge	numericint1eq	numericint1ne

numericint1ne	numericint1gt	numericint1le	numericint1ge	numericint2eq	numericint2ne	numericint2lt
numericint2gt	numericint2le	numericint2ge	numericint4eq	numericint4ne	numericint4lt	numericint4gt
numericint4le	numericint4ge	numericint8eq	numericint8ne	numericint8lt	numericint8gt	numericint8le
numericint8ge	int161eq	int161ne	int161lt	int161gt	int161le	int161ge
int162eq	int162ne	int162lt	int162gt	int162le	int162ge	int164eq
int164ne	int164lt	int164gt	int164le	int164ge	int168eq	int168ne
int168lt	int168gt	int168le	int168ge	int21eq	int21ne	int21lt
int21gt	int21le	int21ge	int216eq	int216ne	int216lt	int216gt
int216le	int216ge	int2numericeq	int2numericne	int2numericlt	int2numericgt	int2numericle
int2numericge	int41eq	int41ne	int41lt	int41gt	int41le	int41ge
int416eq	int416ne	int416lt	int416gt	int416le	int416ge	int4numericeq
int4numericne	int4numericlt	int4numericgt	int4numericle	int4numericge	int81eq	int81ne
int81lt	int81gt	int81le	int81ge	int816eq	int816ne	int816lt
int816gt	int816le	int816ge	int8numericeq	int8numericne	int8numericlt	int8numericgt
bpcharlikebpchar	bpcharnlikebpchar	-	-	-	-	-

NOTE

In upgrade mode, system functions, for example, concat, with variable-length parameters cannot be called.

The following table lists the functions used by GaussDB to implement internal system functions. You are advised not to use these functions. If you need to use them, contact Huawei technical support.

- pv_compute_pool_workload()

  Description: Provides the current load information of the cloud acceleration database instance. (Due to specification changes, the current version no longer supports this feature. Do not use it.)

  Return type: record
- locktag_decode(locktag text)

  Description: Parses lock details from **locktag**.

  Return type: text
- smgreq(a smgr, b smgr)

  Description: Compares two smgrs to check whether they are the same.

  Parameters: smgr, smgr

  Return type: Boolean
- smgrne(a smgr, b smgr)

  Description: Checks whether the two smgrs are different.

  Parameters: smgr, smgr

  Return type: Boolean
- xidin4

  Description: Inputs a 4-byte xid.

  Parameter: cstring

  Return type: xid32
- set_hashbucket_info

  Description: Sets hash bucket information.

  Parameter: text

  Return type: Boolean
- int1send

  Description: Packs unsigned 1-byte integers into the internal data buffer stream.

  Parameter: tinyint

  Return type: bytea
- listagg

  Description: Specifies aggregate functions of the list type.

  Parameters: smallint, text

  Return type: text
- log_fdw_validator

  Description: Specifies validation functions.

  Parameters: text[], oid

  Return type: void
- nvarchar2typmodin

  Description: Obtains the typmod information of the varchar type.

Parameter: cstring[]

Return type: integer

- nvarchar2typmodout

  Description: Obtains the typmod information of the varchar type, constructs a character string, and returns the character string.

  Parameter: integer

  Return type: cstring

- read_disable_conn_file

  Description: Reads forbidden connection files.

  Parameter: nan

  Return type: disconn_mode text, disconn_host text, disconn_port text, local_host text, local_port text, redo_finished text

- regex_like_m

  Description: Specifies the regular expression match, which is used to determine whether a character string complies with a specified regular expression.

  Parameters: text, text

  Return type: Boolean

- update_pgjob

  Description: Updates a job.

  Parameters: bigint, "char", bigint, timestamp without time zone, timestamp without time zone, timestamp without time zone, timestamp without time zone, smallint, text

  Return type: void

- enum_cmp

  Description: Specifies the enumeration comparison function, which is used to determine whether two enumeration classes are equal and determine their relative sizes.

  Parameters: anyenum, anyenum

  Return type: integer

- enum_eq

  Description: Specifies the enumeration comparison function, which is used to implement the equal sign (=).

  Parameters: anyenum, anyenum

  Return type: Boolean

- enum_first

  Description: Returns the first element in the enumeration class.

  Parameter: anyenum

  Return type: anyenum

- enum_ge

  Description: Specifies the enumeration comparison function, which is used to implement the greater-than sign (>) and equal sign (=).

  Parameters: anyenum, anyenum

Return type: Boolean

- enum_gt

  Description: Specifies the enumeration comparison function, which is used to implement the greater-than sign (>).

  Parameters: anyenum, anyenum

  Return type: Boolean

- enum_in

  Description: Specifies the enumeration comparison function, which is used to determine whether an element is in an enumeration class.

  Parameters: cstring, OID

  Return type: anyenum

- enum_larger

  Description: Specifies the enumeration comparison function, which is used to implement the greater-than sign (>).

  Parameters: anyenum, anyenum

  Return type: anyenum

- enum_last

  Description: Returns the last element in the enumeration class.

  Parameter: anyenum

  Return type: anyenum

- enum_le

  Description: Specifies the enumeration comparison function, which is used to implement the less-than sign (<) and equal sign (=).

  Parameters: anyenum, anyenum

  Return type: Boolean

- enum_lt

  Description: Specifies the enumeration comparison function, which is used to implement the less-than sign (<).

  Parameters: anyenum, anyenum

  Return type: Boolean

- enum_smaller

  Description: Specifies the enumeration comparison function, which is used to implement the less-than sign (<).

  Parameters: anyenum, anyenum

  Return type: Boolean

- node_oid_name

  Description: Not supported.

  Parameter: oid

  Return type: cstring

- pg_buffercache_pages

  Description: Reads data from the shared buffer.

  Parameter: nan

Return type: bufferid integer, relfilenode oid, bucketid smallint, reltablespace oid, reldatabase oid, relforknumber smallint, relblocknumber bigint, isdirty boolean, usage_count smallint

- pg_check_xidlimit

  Description: Checks whether nextxid is greater than or equal to xidwarnlimit.

  Parameter: nan

  Return type: Boolean

- gs_static_threadpool_ctrl_status()

  Description: Queries statistics about the static pool threads in the thread pool. The centralized mode does not support the query and returns an error message.

  Parameter: nan

  Return value: text node_name, int group_id, text worker_info

Table 7-152 GS_STATIC_THREADPOOL_CTRL_STATUS columns

Name	Type	Description
node_name	OUT text	Instance name.
group_id	OUT int	Group ID of the thread pool.
worker_info	OUT text	Dynamic statistics about the running thread pool of the current group. The information includes: **default** and **default_s**: number of threads, including default value of the dynamic thread pool and that of the static thread pool. **expect** and **expect_s**: number of threads, including the expected value of the dynamic thread pool and that of the static thread pool. **actual**: actual number of running threads, including dynamic pool threads and static pool threads. **static threads limit**: indicates the number of static pool threads configured for the current group. **has static threads**: specifies whether a static pool is created for the current group. The default value is **0**, indicating no creation. **idle static threads**: number of idle static pool threads. **wait session num**: number of waiting sessions.

- gs_validate_ext_listen_ip

  Description: Clears services connected to invalid IP addresses on DNs.

  Parameter: For details, see **Table 7-153**.

  Return value: bigint pid, text node_name

  Note: When this function is queried in the current environment, the first input parameter can only be set to **'normal'** and no value is returned.

  Table 7-153 GS_VALIDATE_EXT_LISTEN_IP columns

Name	Type	Description
clear	IN cstring	Specifies whether to clear services. **normal** indicates that services connected to invalid IP addresses on DNs are cleared.
validate_node_name	IN cstring	Specifies the name of the DN where the IP address connections to be cleared are located.
validate_ip	IN cstring	Specifies the IP address to be cleared.
pid	OUT bigint	ID of the service thread where the IP address connections to be cleared are located.
node_name	OUT text	Name of the instance where the IP connections to be cleared are located.

- gs_comm_listen_address_ext_info

  Description: Displays the DFX information about the extended IP address configured for **listen_address_ext** connected to the current node.

  Parameter: nan

  Return type: text node_name, text app, bigint tid, integer lwtid, bigint query_id, integer socket, text remote_ip, text remote_port, text local_ip or text local_port

  Note: Currently, this function is not supported.

  Table 7-154 GS_COMM_LISTEN_ADDRESS_EXT_INFO columns

Name	Type	Description
node_name	OUT text	Name of the current instance.
app	OUT text	Client connected to the DN.
tid	OUT bigint	Thread ID of the current thread.
lwtid	OUT integer	Lightweight thread ID of the current thread.
query_id	OUT bigint	Query ID of the current thread.

Name	Type	Description
socket	OUT integer	Socket FD of the current physical connection.
remote_ip	OUT text	Peer IP address of the current connection.
remote_port	OUT text	Peer port of the current connection.
local_ip	OUT text	Local IP address of the current connection.
local_port	OUT text	Local port of the current connection.

- gs_libcomm_fd_info()

  Description: Queries the socket persistent connection information of the libcomm communications library. This function is not supported for centralized.

  Parameter: nan

  Return value: For details, see the following table.

  Table 7-155 Return values of gs_libcomm_fd_info

Name	Type	Description
node_name	OUT text	Instance name of the DN.
ip	OUT text	IP address of the DN.
ctrl_port	OUT bigint	Control channel port.
data_port	OUT bigint	Data channel port.
remote_name	OUT text	Instance name of the DN at the peer site.
remote_ip	OUT text	Peer IP address.
remote_port	OUT bigint	Peer port number.
local_ip	OUT text	Local IP address.
local_port	OUT bigint	Local port number.
fd	OUT bigint	Socket connection.

Name	Type	Description
type	OUT text	Type of the TCP connection channel. The options are as follows: • **DATA_SEND_FD**: socket of the sender of the data channel • **DATA_RECV_FD**: socket of the receiver of the data channel. • **CTL_SEND_FD**: socket of the sender of the control channel. • **CTL_RECV_FD**: socket of the receiver of the control channel.

NOTE

This function can be used only on the DN. If no information is returned, the libcomm persistent connection is not established for the current DN instance.

- gs_libcomm_memory_info()

  Description: Queries the basic configuration information used by the internal memory of the current libcomm communications library. This function is not supported for centralized.

  Parameter: nan

  Return value: For details, see the following table.

  Table 7-156 Return values of gs_libcomm_memory_info

Name	Type	Description
node_name	OUT text	Instance name of the DN.
current_used_memory	OUT bigint	Memory used by the libcomm communications library, in bytes.
current_data_item_num	OUT bigint	Number of data blocks in the libcomm lockless queue.
init_mailbox_memory	OUT bigint	Memory used by libcomm to initialize the mailbox, in bytes.
max_datanode	OUT bigint	Maximum number of DNs supported by the libcomm communications library.
max_stream	OUT bigint	Maximum number of data streams supported by a single TCP connection of the libcomm.
max_quota_size	OUT bigint	Total buffer size of the libcomm logical connection, in bytes.

Name	Type	Description
max_usable_memory	OUT bigint	Maximum memory that can be used by the libcomm communications library cache, in bytes.
max_memory_pool	OUT bigint	Total size of the memory resource pool that can be used by the libcomm communications library on each DN. The unit is byte.

- gs_get_global_listen_address_ext_info

  Description: Queries the global extended IP address configuration information.

  Parameter: For details, see **Table 7-157**.

  Return type: text node_name, text host, text port, text ext_listen_ip

  Note: Currently, this function is not supported.

  Table 7-157 GS_GET_GLOBAL_LISTEN_ADDRESS_EXT_INFO columns

Name	Type	Description
dn_mode	IN cstring	Specifies the range of DNs to be displayed. If this parameter is set to **null**, all DNs are queried by default.
node_name	OUT text	DN name.
host	OUT text	Listening IP address of the DN.
port	OUT text	Listening port of the DN.
ext_listen_ip	OUT text	Extended listening IP address of the DN.

- gs_get_listen_address_ext_info()

  Description: Queries the extended IP address configuration of the current instance.

  Parameter: nan

  Return type: text node_name, text host, bigint port, text ext_listen_ip

  Note: Currently, this function is not supported.

  Table 7-158 GS_GET_LISTEN_ADDRESS_EXT_INFO

Name	Type	Description
node_name	OUT text	DN name.
host	OUT text	Listening IP address of the DN.
port	OUT bigint	Listening port of the DN.

Name	Type	Description
ext_listen_ip	OUT text	Extended listening IP address of the DN.

- pg_comm_delay

  Description: Displays the delay status of the communications library of a single DN.

  Parameter: nan

  Return type: text, text, integer, integer, integer, integer

- pg_comm_recv_stream

  Description: Displays the receiving stream status of all communication libraries on a single DN.

  Parameter: nan

  Return type: text, bigint, text, bigint, integer, integer, integer, text, bigint, integer, integer, integer, bigint, bigint, bigint, bigint, bigint

- pg_comm_send_stream

  Description: Displays the sending stream status of all communication libraries on a single DN.

  Parameter: nan

  Return type: text, bigint, text, bigint, integer, integer, integer, text, bigint, integer, integer, integer, bigint, bigint, bigint, bigint, bigint

- pg_comm_status

  Description: Displays the communication status of a single DN.

  Parameter: nan

  Return type: text, integer, integer, bigint, bigint, bigint, bigint, bigint, integer, integer, integer, integer, integer

- pg_log_comm_status

  Description: Prints some logs on the DN.

  Parameter: nan

  Return type: Boolean

- pg_parse_clog

  Description: Parses Clog to obtain the XID status of an ordinary table.

  Parameter: nan

  Return type: xid xid, status text

- pg_parse_clog(bucketid)

  Description: The hash bucket table is not supported in the current version in centralized mode. An error is reported when the function is called.

- pg_pool_ping

  Description: Sets PoolerPing.

  Parameter: Boolean

  Return type: SETOF boolean

- pg_resume_bkp_flag

  Description: Obtains the delay xlong flag for backup and restoration.

  Parameter: slot_name name

  Return type: start_backup_flag boolean, to_delay boolean, ddl_delay_recycle_ptr text, rewind_time text

- psortoptions

  Description: Returns the psort attribute.

  Parameters: text[], Boolean

  Return type: bytea

- xideq4

  Description: Compares two values of the xid type to check whether they are the same.

  Parameters: xid32, xid32

  Return type: Boolean

- xideqint8

  Description: Compares values of the xid type and int8 type to check whether they are the same.

  Parameters: xid, bigint

  Return type: Boolean

- xidlt

  Description: Returns whether xid1 < xid2 is true.

  Parameters: xid, xid

  Return type: Boolean

- xidlt4

  Description: Returns whether xid1 < xid2 is true.

  Parameters: xid32, xid32

  Return type: Boolean

- gs_shutdown_cross_region_walsenders

  Description: Interrupts cross-cluster streaming replication.

  Parameter: nan

  Return type: void

  Note: To call this function, the user must have the SYSADMIN or OPRADMIN permission, and **operate_mode** must be enabled for the O&M administrator role.

- is_dblink_in_transaction

  Description: Checks whether a database link corresponding to an OID is used in the current transaction.

  Parameter: oid

  Return type: Boolean

- dblink_has_updatasent

  Description: Checks whether DML statements are sent using a database link corresponding to an OID in the current transaction and are not committed.

Parameter: oid

Return type: Boolean

- get_last_xmin_by_oid

  Description: Obtains the maximum **xmin** value of all columns in a table based on the table OID.

  Parameter: oid

  Return type: xid

- get_relid_by_relname

  Description: Obtains the OID of a table based on the table name and relnamespace.

  Parameters: cstring, OID

  Return type: oid

- copy_summary_create

  Description: Creates a gs_copy_summary table.

  Parameter: none

  Return type: text

- btint12cmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to another one, **0** is returned. If the value is less than another one, a negative number is returned.

  Parameters: int1, int2

  Return type: integer

- btint14cmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to another one, **0** is returned. If the value is less than another one, a negative number is returned.

  Parameters: int1, int4

  Return type: integer

- btint18cmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to another one, **0** is returned. If the value is less than another one, a negative number is returned.

  Parameters: int1, int8

  Return type: integer

- btint116cmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to another one, **0** is returned. If the value is less than another one, a negative number is returned.

  Parameters: int1, int16

  Return type: integer

- btint1numericcmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to another one, **0** is returned. If the value is less than another one, a negative number is returned.

  Parameters: int1, numeric

  Return type: integer

- btint21cmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to another one, **0** is returned. If the value is less than another one, a negative number is returned.

  Parameters: int2, int1

  Return type: integer

- btint216cmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to another one, **0** is returned. If the value is less than another one, a negative number is returned.

  Parameters: int2, int16

  Return type: integer

- btint2numericcmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to another one, **0** is returned. If the value is less than another one, a negative number is returned.

  Parameters: int2, numeric

  Return type: integer

- btint41cmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to another one, **0** is returned. If the value is less than another one, a negative number is returned.

  Parameters: int4, int1

  Return type: integer

- btint416cmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to another one, **0** is returned. If the value is less than another one, a negative number is returned.

  Parameters: int4, int16

  Return type: integer

- btint4numericcmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to

another one, **0** is returned. If the value is less than another one, a negative number is returned.

Parameters: int4, numeric

Return type: integer

- btint81cmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to another one, **0** is returned. If the value is less than another one, a negative number is returned.

  Parameters: int8, int1

  Return type: integer

- btint816cmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to another one, **0** is returned. If the value is less than another one, a negative number is returned.

  Parameters: int8, int16

  Return type: integer

- btint8numericcmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to another one, **0** is returned. If the value is less than another one, a negative number is returned.

  Parameters: int8, numeric

  Return type: integer

- btint161cmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to another one, **0** is returned. If the value is less than another one, a negative number is returned.

  Parameters: int16, int1

  Return type: integer

- btint162cmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to another one, **0** is returned. If the value is less than another one, a negative number is returned.

  Parameters: int16, int2

  Return type: integer

- btint164cmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to another one, **0** is returned. If the value is less than another one, a negative number is returned.

  Parameters: int16, int4

Return type: integer

- btint168cmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to another one, **0** is returned. If the value is less than another one, a negative number is returned.

  Parameters: int16, int8

  Return type: integer

- btnumericint1cmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to another one, **0** is returned. If the value is less than another one, a negative number is returned.

  Parameters: numeric, int1

  Return type: integer

- btnumericint2cmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to another one, **0** is returned. If the value is less than another one, a negative number is returned.

  Parameters: numeric, int2

  Return type: integer

- btnumericint4cmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to another one, **0** is returned. If the value is less than another one, a negative number is returned.

  Parameters: numeric, int4

  Return type: integer

- btnumericint8cmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to another one, **0** is returned. If the value is less than another one, a negative number is returned.

  Parameters: numeric, int8

  Return type: integer

- btint16cmp()

  Description: Compares the values of two parameters. If the value is greater than another one, a positive number is returned. If the value is equal to another one, **0** is returned. If the value is less than another one, a negative number is returned.

  Parameters: int16, int16

  Return type: integer

- hashint16()

  Description: Calculates the hash value of an input parameter.

Parameter: int16

Return type: integer

- hashint1_numeric()

  Description: Calculates the hash value of an input parameter.

  Parameter: int1

  Return type: integer

- hashint2_numeric()

  Description: Calculates the hash value of an input parameter.

  Parameter: int2

  Return type: integer

- hashint4_numeric()

  Description: Calculates the hash value of an input parameter.

  Parameter: int4

  Return type: integer

- hashint8_numeric()

  Description: Calculates the hash value of an input parameter.

  Parameter: int8

  Return type: integer

- int12eq()

  Description: Compares whether two parameter values are equal. If they are equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int2

  Return type: Boolean

- int14eq()

  Description: Compares whether two parameter values are equal. If they are equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int4

  Return type: Boolean

- int18eq()

  Description: Compares whether two parameter values are equal. If they are equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int8

  Return type: Boolean

- int116eq()

  Description: Compares whether two parameter values are equal. If they are equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int16

  Return type: Boolean

- int1numericeq()

  Description: Compares whether two parameter values are equal. If they are equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, numeric

Return type: Boolean

- int21eq()

  Description: Compares whether two parameter values are equal. If they are equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int2, int1

  Return type: Boolean

- int216eq()

  Description: Compares whether two parameter values are equal. If they are equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int2, int16

  Return type: Boolean

- int2numericeq()

  Description: Compares whether two parameter values are equal. If they are equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int2, numeric

  Return type: Boolean

- int41eq()

  Description: Compares whether two parameter values are equal. If they are equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int4, int1

  Return type: Boolean

- int416eq()

  Description: Compares whether two parameter values are equal. If they are equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int4, int16

  Return type: Boolean

- int4numericeq()

  Description: Compares whether two parameter values are equal. If they are equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int4, numeric

  Return type: Boolean

- int81eq()

  Description: Compares whether two parameter values are equal. If they are equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int8, int1

  Return type: Boolean

- int816eq()

  Description: Compares whether two parameter values are equal. If they are equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int8, int16

  Return type: Boolean

- int8numericeq()

Description: Compares whether two parameter values are equal. If they are equal, **true** is returned. Otherwise, **false** is returned.

Parameters: int8, numeric

Return type: Boolean

- int161eq()

  Description: Compares whether two parameter values are equal. If they are equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int16, int1

  Return type: Boolean

- int162eq()

  Description: Compares whether two parameter values are equal. If they are equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int16, int2

  Return type: Boolean

- int164eq()

  Description: Compares whether two parameter values are equal. If they are equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int16, int4

  Return type: Boolean

- int168eq()

  Description: Compares whether two parameter values are equal. If they are equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int16, int8

  Return type: Boolean

- numericint1eq()

  Description: Compares whether two parameter values are equal. If they are equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int1

  Return type: Boolean

- numericint2eq()

  Description: Compares whether two parameter values are equal. If they are equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int2

  Return type: Boolean

- numericint4eq()

  Description: Compares whether two parameter values are equal. If they are equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int4

  Return type: Boolean

- numericint8eq()

  Description: Compares whether two parameter values are equal. If they are equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int8

Return type: Boolean

- int12ne()

  Description: Compares whether two parameter values are equal. If they are not equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int2

  Return type: Boolean

- int14ne()

  Description: Compares whether two parameter values are equal. If they are not equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int4

  Return type: Boolean

- int18ne()

  Description: Compares whether two parameter values are equal. If they are not equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int8

  Return type: Boolean

- int116ne()

  Description: Compares whether two parameter values are equal. If they are not equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int16

  Return type: Boolean

- int1numericne()

  Description: Compares whether two parameter values are equal. If they are not equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, numeric

  Return type: Boolean

- int21ne()

  Description: Compares whether two parameter values are equal. If they are not equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int2, int1

  Return type: Boolean

- int216ne()

  Description: Compares whether two parameter values are equal. If they are not equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int2, int16

  Return type: Boolean

- int2numericne()

  Description: Compares whether two parameter values are equal. If they are not equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int2, numeric

  Return type: Boolean

- int41ne()

Description: Compares whether two parameter values are equal. If they are not equal, **true** is returned. Otherwise, **false** is returned.

Parameters: int4, int1

Return type: Boolean

- int416ne()

  Description: Compares whether two parameter values are equal. If they are not equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int4, int16

  Return type: Boolean

- int4numericne()

  Description: Compares whether two parameter values are equal. If they are not equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int4, numeric

  Return type: Boolean

- int81ne()

  Description: Compares whether two parameter values are equal. If they are not equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int8, int1

  Return type: Boolean

- int816ne()

  Description: Compares whether two parameter values are equal. If they are not equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int8, int16

  Return type: Boolean

- int8numericne()

  Description: Compares whether two parameter values are equal. If they are not equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int8, numeric

  Return type: Boolean

- int161ne()

  Description: Compares whether two parameter values are equal. If they are not equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int16, int1

  Return type: Boolean

- int162ne()

  Description: Compares whether two parameter values are equal. If they are not equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int16, int2

  Return type: Boolean

- int164ne()

  Description: Compares whether two parameter values are equal. If they are not equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int16, int4

Return type: Boolean

- int168ne()

  Description: Compares whether two parameter values are equal. If they are not equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: int16, int8

  Return type: Boolean

- numericint1ne()

  Description: Compares whether two parameter values are equal. If they are not equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int1

  Return type: Boolean

- numericint2ne()

  Description: Compares whether two parameter values are equal. If they are not equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int2

  Return type: Boolean

- numericint4ne()

  Description: Compares whether two parameter values are equal. If they are not equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int4

  Return type: Boolean

- numericint8ne()

  Description: Compares whether two parameter values are equal. If they are not equal, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int8

  Return type: Boolean

- int12lt()

  Description: Compares whether a parameter is less than another one. If the parameter is less than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int2

  Return type: Boolean

- int14lt()

  Description: Compares whether a parameter is less than another one. If the parameter is less than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int4

  Return type: Boolean

- int18lt()

  Description: Compares whether a parameter is less than another one. If the parameter is less than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int8

Return type: Boolean

- int116lt()

  Description: Compares whether a parameter is less than another one. If the parameter is less than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int16

  Return type: Boolean

- int1numericlt()

  Description: Compares whether a parameter is less than another one. If the parameter is less than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, numeric

  Return type: Boolean

- int21lt()

  Description: Compares whether a parameter is less than another one. If the parameter is less than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int2, int1

  Return type: Boolean

- int216lt()

  Description: Compares whether a parameter is less than another one. If the parameter is less than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int2, int16

  Return type: Boolean

- int2numericlt()

  Description: Compares whether a parameter is less than another one. If the parameter is less than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int2, numeric

  Return type: Boolean

- int41lt()

  Description: Compares whether a parameter is less than another one. If the parameter is less than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int4, int1

  Return type: Boolean

- int416lt()

  Description: Compares whether a parameter is less than another one. If the parameter is less than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int4, int16

  Return type: Boolean

- int4numericlt()

Description: Compares whether a parameter is less than another one. If the parameter is less than another one, **true** is returned. Otherwise, **false** is returned.

Parameters: int4, numeric

Return type: Boolean

- int81lt()

  Description: Compares whether a parameter is less than another one. If the parameter is less than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int8, int1

  Return type: Boolean

- int816lt()

  Description: Compares whether a parameter is less than another one. If the parameter is less than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int8, int16

  Return type: Boolean

- int8numericlt()

  Description: Compares whether a parameter is less than another one. If the parameter is less than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int8, numeric

  Return type: Boolean

- int161lt()

  Description: Compares whether a parameter is less than another one. If the parameter is less than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int16, int1

  Return type: Boolean

- int162lt()

  Description: Compares whether a parameter is less than another one. If the parameter is less than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int16, int2

  Return type: Boolean

- int164lt()

  Description: Compares whether a parameter is less than another one. If the parameter is less than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int16, int4

  Return type: Boolean

- int168lt()

  Description: Compares whether a parameter is less than another one. If the parameter is less than another one, **true** is returned. Otherwise, **false** is returned.

Parameters: int16, int8

Return type: Boolean

- numericint1lt()

  Description: Compares whether a parameter is less than another one. If the parameter is less than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int1

  Return type: Boolean

- numericint2lt()

  Description: Compares whether a parameter is less than another one. If the parameter is less than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int2

  Return type: Boolean

- numericint4lt()

  Description: Compares whether a parameter is less than another one. If the parameter is less than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int4

  Return type: Boolean

- numericint8lt()

  Description: Compares whether a parameter is less than another one. If the parameter is less than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int8

  Return type: Boolean

- int12gt()

  Description: Compares whether a parameter is greater than another one. If the parameter is greater than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int2

  Return type: Boolean

- int14gt()

  Description: Compares whether a parameter is greater than another one. If the parameter is greater than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int4

  Return type: Boolean

- int18gt()

  Description: Compares whether a parameter is greater than another one. If the parameter is greater than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int8

  Return type: Boolean

- int116gt()

  Description: Compares whether a parameter is greater than another one. If the parameter is greater than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int16

  Return type: Boolean

- int1numericgt()

  Description: Compares whether a parameter is greater than another one. If the parameter is greater than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, numeric

  Return type: Boolean

- int21gt()

  Description: Compares whether a parameter is greater than another one. If the parameter is greater than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int2, int1

  Return type: Boolean

- int216gt()

  Description: Compares whether a parameter is greater than another one. If the parameter is greater than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int2, int16

  Return type: Boolean

- int2numericgt()

  Description: Compares whether a parameter is greater than another one. If the parameter is greater than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int2, numeric

  Return type: Boolean

- int41gt()

  Description: Compares whether a parameter is greater than another one. If the parameter is greater than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int4, int1

  Return type: Boolean

- int416gt()

  Description: Compares whether a parameter is greater than another one. If the parameter is greater than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int4, int16

  Return type: Boolean

- int4numericgt()

Description: Compares whether a parameter is greater than another one. If the parameter is greater than another one, **true** is returned. Otherwise, **false** is returned.

Parameters: int4, numeric

Return type: Boolean

- int81gt()

  Description: Compares whether a parameter is greater than another one. If the parameter is greater than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int8, int1

  Return type: Boolean

- int816gt()

  Description: Compares whether a parameter is greater than another one. If the parameter is greater than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int8, int16

  Return type: Boolean

- int8numericgt()

  Description: Compares whether a parameter is greater than another one. If the parameter is greater than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int8, numeric

  Return type: Boolean

- int161gt()

  Description: Compares whether a parameter is greater than another one. If the parameter is greater than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int16, int1

  Return type: Boolean

- int162gt()

  Description: Compares whether a parameter is greater than another one. If the parameter is greater than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int16, int2

  Return type: Boolean

- int164gt()

  Description: Compares whether a parameter is greater than another one. If the parameter is greater than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int16, int4

  Return type: Boolean

- int168gt()

  Description: Compares whether a parameter is greater than another one. If the parameter is greater than another one, **true** is returned. Otherwise, **false** is returned.

Parameters: int16, int8

Return type: Boolean

- numericint1gt()

  Description: Compares whether a parameter is greater than another one. If the parameter is greater than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int1

  Return type: Boolean

- numericint2gt()

  Description: Compares whether a parameter is greater than another one. If the parameter is greater than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int2

  Return type: Boolean

- numericint4gt()

  Description: Compares whether a parameter is greater than another one. If the parameter is greater than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int4

  Return type: Boolean

- numericint8gt()

  Description: Compares whether a parameter is greater than another one. If the parameter is greater than another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int8

  Return type: Boolean

- int12le()

  Description: Compares whether a parameter is less than or equal to another one. If the parameter is less than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int2

  Return type: Boolean

- int14le()

  Description: Compares whether a parameter is less than or equal to another one. If the parameter is less than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int4

  Return type: Boolean

- int18le()

  Description: Compares whether a parameter is less than or equal to another one. If the parameter is less than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int8

  Return type: Boolean

- int116le()

  Description: Compares whether a parameter is less than or equal to another one. If the parameter is less than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int16

  Return type: Boolean

- int1numericle()

  Description: Compares whether a parameter is less than or equal to another one. If the parameter is less than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, numeric

  Return type: Boolean

- int21le()

  Description: Compares whether a parameter is less than or equal to another one. If the parameter is less than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int2, int1

  Return type: Boolean

- int216le()

  Description: Compares whether a parameter is less than or equal to another one. If the parameter is less than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int2, int16

  Return type: Boolean

- int2numericle()

  Description: Compares whether a parameter is less than or equal to another one. If the parameter is less than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int2, numeric

  Return type: Boolean

- int41le()

  Description: Compares whether a parameter is less than or equal to another one. If the parameter is less than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int4, int1

  Return type: Boolean

- int416le()

  Description: Compares whether a parameter is less than or equal to another one. If the parameter is less than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int4, int16

  Return type: Boolean

- int4numericle()

Description: Compares whether a parameter is less than or equal to another one. If the parameter is less than or equal to another one, **true** is returned. Otherwise, **false** is returned.

Parameters: int4, numeric

Return type: Boolean

- int81le()

  Description: Compares whether a parameter is less than or equal to another one. If the parameter is less than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int8, int1

  Return type: Boolean

- int816le()

  Description: Compares whether a parameter is less than or equal to another one. If the parameter is less than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int8, int16

  Return type: Boolean

- int8numericle()

  Description: Compares whether a parameter is less than or equal to another one. If the parameter is less than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int8, numeric

  Return type: Boolean

- int161le()

  Description: Compares whether a parameter is less than or equal to another one. If the parameter is less than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int16, int1

  Return type: Boolean

- int162le()

  Description: Compares whether a parameter is less than or equal to another one. If the parameter is less than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int16, int2

  Return type: Boolean

- int164le()

  Description: Compares whether a parameter is less than or equal to another one. If the parameter is less than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int16, int4

  Return type: Boolean

- int168le()

  Description: Compares whether a parameter is less than or equal to another one. If the parameter is less than or equal to another one, **true** is returned. Otherwise, **false** is returned.

Parameters: int16, int8

Return type: Boolean

- numericint1le()

  Description: Compares whether a parameter is less than or equal to another one. If the parameter is less than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int1

  Return type: Boolean

- numericint2le()

  Description: Compares whether a parameter is less than or equal to another one. If the parameter is less than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int2

  Return type: Boolean

- numericint4le()

  Description: Compares whether a parameter is less than or equal to another one. If the parameter is less than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int4

  Return type: Boolean

- numericint8le()

  Description: Compares whether a parameter is less than or equal to another one. If the parameter is less than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int8

  Return type: Boolean

- int12ge()

  Description: Compares whether a parameter is greater than or equal to another one. If the parameter is greater than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int2

  Return type: Boolean

- int14ge()

  Description: Compares whether a parameter is greater than or equal to another one. If the parameter is greater than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int4

  Return type: Boolean

- int18ge()

  Description: Compares whether a parameter is greater than or equal to another one. If the parameter is greater than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int8

  Return type: Boolean

- int116ge()

  Description: Compares whether a parameter is greater than or equal to another one. If the parameter is greater than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, int16

  Return type: Boolean

- int1numericge()

  Description: Compares whether a parameter is greater than or equal to another one. If the parameter is greater than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int1, numeric

  Return type: Boolean

- int21ge()

  Description: Compares whether a parameter is greater than or equal to another one. If the parameter is greater than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int2, int1

  Return type: Boolean

- int216ge()

  Description: Compares whether a parameter is greater than or equal to another one. If the parameter is greater than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int2, int16

  Return type: Boolean

- int2numericge()

  Description: Compares whether a parameter is greater than or equal to another one. If the parameter is greater than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int2, numeric

  Return type: Boolean

- int41ge()

  Description: Compares whether a parameter is greater than or equal to another one. If the parameter is greater than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int4, int1

  Return type: Boolean

- int416ge()

  Description: Compares whether a parameter is greater than or equal to another one. If the parameter is greater than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int4, int16

  Return type: Boolean

- int4numericge()

Description: Compares whether a parameter is greater than or equal to another one. If the parameter is greater than or equal to another one, **true** is returned. Otherwise, **false** is returned.

Parameters: int4, numeric

Return type: Boolean

- int81ge()

  Description: Compares whether a parameter is greater than or equal to another one. If the parameter is greater than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int8, int1

  Return type: Boolean

- int816ge()

  Description: Compares whether a parameter is greater than or equal to another one. If the parameter is greater than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int8, int16

  Return type: Boolean

- int8numericge()

  Description: Compares whether a parameter is greater than or equal to another one. If the parameter is greater than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int8, numeric

  Return type: Boolean

- int161ge()

  Description: Compares whether a parameter is greater than or equal to another one. If the parameter is greater than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int16, int1

  Return type: Boolean

- int162ge()

  Description: Compares whether a parameter is greater than or equal to another one. If the parameter is greater than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int16, int2

  Return type: Boolean

- int164ge()

  Description: Compares whether a parameter is greater than or equal to another one. If the parameter is greater than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: int16, int4

  Return type: Boolean

- int168ge()

  Description: Compares whether a parameter is greater than or equal to another one. If the parameter is greater than or equal to another one, **true** is returned. Otherwise, **false** is returned.

Parameters: int16, int8

Return type: Boolean

- numericint1ge()

  Description: Compares whether a parameter is greater than or equal to another one. If the parameter is greater than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int1

  Return type: Boolean

- numericint2ge()

  Description: Compares whether a parameter is greater than or equal to another one. If the parameter is greater than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int2

  Return type: Boolean

- numericint4ge()

  Description: Compares whether a parameter is greater than or equal to another one. If the parameter is greater than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int4

  Return type: Boolean

- numericint8ge()

  Description: Compares whether a parameter is greater than or equal to another one. If the parameter is greater than or equal to another one, **true** is returned. Otherwise, **false** is returned.

  Parameters: numeric, int8

  Return type: Boolean

- gs_relation_is_updatable()

  Description: Returns whether a relationship can be inserted, updated, or deleted. If the value can be updated, **4** is returned. If the value can be inserted, **8** is returned. If the value can be deleted, **16** is returned. If the value meets the requirements, the value obtained by adding the corresponding values is returned.

  Parameters: OID, Boolean

  Return type: int32

- gs_column_is_updatable()

  Description: Returns whether a column can be updated. If the column can be updated, **t** is returned. Otherwise, **f** is returned.

  Parameters: OID, int16, Boolean

  Return type: Boolean

7.6.39 Internal Functions

The following functions of GaussDB use internal data types, which cannot be directly called by users:

- Selectivity calculation functions

areajoinsel	areasel	arraycontjoinsel	arraycontsel	contjoinsel	contsel	eqjoinsel
eqsel	iclikejoinsel	iclikesel	icnlikejoinsel	icnlikesel	icregexeqjoinsel	icregexeqsel
icregexnejoinsel	icregexnesel	likejoinsel	likesel	neqjoinsel	neqsel	nlikejoinsel
nlikesel	positionjoinsel	positionsel	regexeqjoinsel	regexeqsel	regexnejoinsel	regexnesel
scalargtjoinsel	scalargtsel	scalarltjoinsel	scalarltsel	tsmatchjoinsel	tsmatchsel	-

- Statistics collection functions

array_typanalyze	range_typanalyze	ts_typanalyze
local_rto_stat	-	-

- Internal functions for sorting

bpchar_sortsupport	bytea_sortsupport	date_sortsupport	numeric_sortsupport	timestamp_sortsupport

- Internal type processing functions

abstimerecv	euc_jis_2004_to_utf8	int2recv	line_recv	oidvectorrecv_extend	tidrecv	utf8_to_koi8u
anyarray_recv	euc_jp_to_mic	int2vectorrecv	lseg_recv	path_recv	time_recv	utf8_to_shift_jis_2004
array_recv	euc_jp_to_sjis	int4recv	macaddr_recv	pg_node_tree_recv	time_transform	utf8_to_sjis
ascii_to_mic	euc_jp_to_utf8	int8recv	mic_to_ascii	point_recv	timestamp_recv	utf8_to_uhc
ascii_to_utf8	euc_kr_to_mic	internal_out	mic_to_big5	poly_recv	timestamp_transform	utf8_to_win
big5_to_euc_tw	euc_kr_to_utf8	interval_recv	mic_to_euc_cn	pound_nexttoken	timestamptz_recv	uuid_recv

big5_to_mic	euc_tw_to_big5	interval_transform	mic_to_euc_jp	prsd_nexttoken	timetz_recv	varbit_recv
big5_to_utf8	euc_tw_to_mic	iso_to_koi8r	mic_to_euc_kr	range_recv	tinterval recv	varbit_transform
bit_recv	euc_tw_to_utf8	iso_to_mic	mic_to_euc_tw	rawrecv	tsqueryrecv	varchar_transform
boolrecv	float4recv	iso_to_win1251	mic_to_iso	record_recv	tsvectorrecv	varcharrecv
box_recv	float8recv	iso_to_win866	mic_to_koi8r	regclassrecv	txid_snapshot_recv	void_recv
bpcharrecv	gb18030_to_utf8	iso8859_1_to_utf8	mic_to_latin1	regconfigrecv	uhc_to_utf8	win_to_utf8
btoidsortsupport	gbk_to_utf8	iso8859_to_utf8	mic_to_latin2	regdictionaryrecv	unknownrecv	win1250_to_latin2
byteavecv	-	johab_to_utf8	mic_to_latin3	regoperatorrecv	utf8_to_ascii	win1250_to_mic
byteawithoutorderwithequalcolrecv	gtsvector_compress	json_recv	mic_to_latin4	regoperrecv	utf8_to_big5	win1251_to_iso
cash_recv	gtsvector_consistent	koi8r_to_iso	mic_to_sjis	regprocedurerecv	utf8_to_euc_cn	win1251_to_koi8r
charrecv	gtsvector_decompress	koi8r_to_mic	mic_to_win1250	regprocrecv	utf8_to_euc_jis_2004	win1251_to_mic
cidr_recv	gtsvector_penalty	koi8r_to_utf8	mic_to_win1251	regtyperecv	utf8_to_euc_jp	win1251_to_win866
cidrecv	gtsvector_picksplit	koi8r_to_win1251	mic_to_win866	reltimerecv	utf8_to_euc_kr	win866_to_iso
circle_recv	gtsvector_same	koi8r_to_win866	namerecv	shift_jis_2004_to_euc_jis_2004	utf8_to_euc_tw	win866_to_koi8r

cstring_recv	gtsvector_union	koi8u_to_utf8	ngram_nexttoken	shift_jis_2004_to_utf8	utf8_to_gb18030	win866_to_mic
date_recv	hll_recv	latin1_to_mic	numeric_recv	sjis_to_euc_jp	utf8_to_gbk	win866_to_win1251
domain_recv	hll_trans_recv	latin2_to_mic	numeric_transform	sjis_to_mic	utf8_to_iso8859	xidrecv
euc_cn_to_mic	-	latin2_to_win1250	nvarchar2recv	sjis_to_utf8	utf8_to_iso8859_1	xidrecv4
euc_cn_to_utf8	inet_recv	latin3_to_mic	oidrecv	smalldatetime_recv	utf8_to_johab	xml_recv
euc_jis_2004_to_shift_jis_2004	int1recv	latin4_to_mic	oidvectorrecv	textrecv	utf8_to_koi8r	-
i16toi1	int16	int16_bool	int16eq	int16div	int16ge	int16gt
int16in	int16le	int16lt	int16mi	int16mul	int16ne	int16out
int16pl	int16recv	int16send	numeric_bool	int2vectorin_extend	int2vectorout_extend	int2vectorrecv_extend
int2vectorsend_extend	tdigest_in	tdigest_merge	tdigest_merge_to_one	tdigest_mergep	tdigest_out	-
anyset_out	btint2setcmp	btint4setcmp	btint8setcmp	btsetcmp	btsetint2cmp	btsetint4cmp
btsetint8cmp	btsetsortsupport	float4	float8	hashsetint	hashsettext	int2
int2seteq	int2setge	int2setgt	int2setle	int2setlt	int2setne	int4
int4seteq	int4setge	int4setgt	int4setle	int4setlt	int4setne	int8
int8seteq	int8setge	int8setgt	int8setle	int8setlt	int8setne	set
set_in	set_out	set_recv	set_send	seteq	setge	setgt
setint2eq	setint2ge	setint2gt	setint2le	setint2lt	setint2ne	setint4eq

setint4ge	setint4gt	setint4le	setint4lt	setint4ne	setint8eq	setint8ge
setint8gt	setint8le	setint8lt	setint8ne	setle	setlt	setne
settexteq	settextge	settextgt	settextle	settextlt	settextne	settobpchar
settonumber	settonvarchar2	settotext	settovarchar	textseteq	textsetge	textsetgt
textsetle	textsetlt	textsetne	gb18030_2022_to_utf8	utf8_to_gb18030_2022	array_to_nesttable	array_to_indexby_int_table
nesttable_to_array	indexbytableint_to_array	array_to_nesttable	array_to_indexby_int_table	zhs16gbk_to_utf8	utf8_to_zhs16gbk	zhs16gbk_to_gb18030
gb18030_to_zhs16gbk	zhs16gbk_to_gb18030_2022	gb18030_2022_to_zhs16gbk	-	-	-	-

- Internal functions for aggregation operations

array_agg_finalfn	array_agg_transfn	bytea_string_agg_finalfn	bytea_string_agg_transfn	date_list_agg_noarg2_transfn	date_list_agg_transfn	float4_list_agg_noarg2_transfn
float4_list_agg_transfn	float8_list_agg_noarg2_transfn	float8_list_agg_transfn	int2_list_agg_noarg2_transfn	int2_list_agg_transfn	int4_list_agg_noarg2_transfn	int4_list_agg_transfn
int8_list_agg_noarg2_transfn	int8_list_agg_transfn	interval_list_agg_noarg2_transfn	interval_list_agg_transfn	list_agg_finalfn	list_agg_noarg2_transfn	list_agg_transfn
median_float8_finalfn	median_interval_finalfn	median_transfn	mode_final	numeric_list_agg_noarg2_transfn	numeric_list_agg_transfn	ordered_set_transition
percentile_cont_float8_final	percentile_cont_interval_final	string_agg_finalfn	string_agg_transfn	timestamp_list_agg_noarg2_transfn	timestamp_list_agg_transfn	timestamptz_list_agg_noarg2_transfn

timestamptz_list_agg_transfn	checksumtext_agg_transfn	-	-	-	-	-

- Hash internal functions

hashbeginscan	hashbuild	hashbuildempty	hashbulkdelete	hashcostestimate	hashendscan	hashgetbitmap
hashgettuple	hashinsert	hashmarkpos	hashmerge	hashrescan	hashrestrpos	hashvacuumcleanup
hashvarlena	-	-	-	-	-	-

- Internal functions of the B-tree index

cbtreebuild	cbtreecanreturn	cbtreecostestimate	cbtreegetbitmap	cbtreegettuple	btbeginscan	btbuild
btbuildempty	btbulkdelete	btcanreturn	btcostestimate	btendscan	btfloat4sortsupport	btfloat8sortsupport
btgetbitmap	btgettuple	btinsert	btint2sortsupport	btint4sortsupport	btint8sortsupport	btmarkpos
btmerge	btnamesortsupport	btrescan	btrestrpos	bttextsortsupport	btvacuumcleanup	cbtreeoptions

- Internal functions of the Psort index

psortbuild	psortcanreturn	psortcostestimate	psortgetbitmap	psortgettuple

- Internal functions of the UBTree index

ubtbeginscan	ubtbuild	ubtbuildempty	ubtbulkdelete	ubtcanreturn
ubtcostestimate	ubtendscan	ubtgetbitmap	ubtgettuple	ubtinsert
ubtmarkpos	ubtmerge	ubtoptions	ubtrescan	ubtrestrpos
ubtvacuumcleanup	-	-	-	-

- plpgsql internal function

  plpgsql_inline_handler

- Internal functions related to collections

array_indexby_delete	array_indexby_length	array_integer_deleteidx	array_integer_exists	array_integer_first	array_integer_last
array_integer_next	array_integer_prior	array_varchar_deleteidx	array_varchar_exists	array_varchar_first	array_varchar_last
array_varchar_next	array_varchar_prior	-	-	-	-

- External table-related internal functions

dist_fdw_handler	roach_handler	streaming_fdw_handler	dist_fdw_validator	file_fdw_handler	file_fdw_validator	log_fdw_handler
dblink_fdw_handler	dblink_fdw_validator	-	-	-	-	-

- Auxiliary functions for the primary DN to remotely read the data page from the standby DN.

  gs_read_block_from_remote is used to read pages of a non-segment-page table file. By default, only the initial user can view the data. Other users can use the data only after being granted with permissions.

  gs_read_segment_block_from_remote is used to read pages of a segment-page table file. By default, only the initial user can view the data. Other users can use the data only after being granted with permissions.

- Auxiliary functions for the primary DN to remotely read data files from the standby DN.

  gs_read_file_from_remote is used to read a specified file. gs_repair_file uses this function to read the remote file segment by segment after using the gs_read_file_size_from_remote function to obtain the file size. By default, only the initial user can view the data. Other users can use the data only after being granted with permissions.

  gs_read_file_size_from_remote is used to read the size of a specified file. Before using the gs_repair_file function to repair a file, you need to obtain the size of the file from the remote end to verify the missing file information and repair the missing files one by one. By default, only the initial user can view the data. Other users can use the data only after being granted with permissions.

- Auxiliary functions for incrementally rebuilding other standby or cascaded standby DNs using the standby DN.

  gs_standby_incremental_filemap_create is used to create a temporary filemap file for incremental rebuilding on the standby DN. The filemap file is used to store the path and size of the data to be transferred during the incremental rebuilding. This API can be called only when the initial user is used and **application** is **gs_rewind**.

gs_standby_incremental_filemap_insert is used to insert file information into a specified temporary filemap file. The file information includes the file path, transfer start point, length of data to be transferred at a time, and rebuild flag bit. This API can be called only when the initial user is used and **application** is **gs_rewind**.

gs_standby_incremental_filemap_execute is used to obtain file information stored in a specified temporary filemap file and delete the specified filemap file for data transmission during incremental standby DN rebuilding. This API can be called only when the initial user is used and **application** is **gs_rewind**.

- Ledger database function

  get_dn_hist_relhash

- AI feature functions

create_snapshot	create_snapshot_internal	prepare_snapshot_internal	prepare_snapshot	manage_snapshot_internal	archive_snapshot	publish_snapshot
purge_snapshot_internal	purge_snapshot	sample_snapshot	-	-	-	-

  – PKG_SERVICE functions

isubmit_on_nodes	submit_on_nodes	-	-	-	-	-

- Other functions

to_tsvector_for_batch	value_of_percentile	disable_conn	bind_variable	job_update	job_cancel	job_finish
similar_escape	table_skewness (unavailable)	timetz_text	time_text	reltime_text	abstime_text	_pg_keysequal
analyze_query (unavailable)	analyze_workload (unavailable)	ssign_table_type	gs_comm_proxy_thread_status	gs_txid_oldestxmin	-	pg_stat_segment_space_info
remote_segment_space_info	set_cost_params	set_weight_params	start_collect_workload	tdigest_in	tdigest_merge	tdigest_merge_to_one

tdigest_mergep	tdigest_out	-	-	-	-	-

- View-related reference functions

  adm_hist_sqlstat_func

  adm_hist_sqlstat_idlog_func

  adm_hist_sqltext_func

- gs_txn_snapshot system catalog maintenance function

  gs_insert_delete_txn_snapshot is used to maintain the gs_txn_snapshot system catalog of each global node in GTM-lite mode. Only the system administrator can call this function. In the current version, if this function is called, **f** is returned.

- XMLType function

  isschemavalid

7.6.40 Global SysCache Functions

- gs_gsc_table_detail(database_id default NULL, rel_id default NULL)

  Description: Queries the table metadata in the global system cache in a database. The user who calls this function must have the SYSADMIN permission.

  Parameter: Specifies the database and table whose global system cache is to be queried. The default value of **database_id** is **NULL** or **–1**, indicating all databases. The value **0** indicates a shared table. Other values indicate the specified database and shared table. **rel_id** indicates the OID of the specified table. The default value **NULL** or **–1** indicates all tables. Other values indicate the specified table. If **database_id** does not exist, an error is reported. If **rel_id** does not exist, a null result is returned.

  Return type: tuple

  Example:

```
SELECT * FROM gs_gsc_table_detail(-1) limit 1;
database_oid | database_name | reloid |          relname          | relnamespace | reltype | reloftype |
relowner | relam | relfilenode | reltablespace | relhasindex | relisshared | relkind | relnatts | relhasoids |
relhaspkey | parttype | tdhasuids | attnames  | extinfo
--------------+---------------+--------+---------------------------+--------------+---------+-----------+----------
+-------+-------------+---------------+-------------+-------------+---------+----------+------------+----------
+----------+-----------+-----------+---------
0 |               |   2676 | pg_authid_rolname_index |           11 |       0 |         0 |       10 |   403 |           0
|         1664 | f           | t           | i       |         1 | f           | f           | n       | f         | 'rolname' |
(1 row)
```

- gs_gsc_catalog_detail(database_id default NULL, rel_id default NULL)

  Description: Queries the system catalog row information in the global system cache in a database. The user who calls this function must have the SYSADMIN permission.

  Parameter: Specifies the database and table whose global system cache is to be queried. The default value of **database_id** is **NULL** or **–1**, indicating all databases. The value **0** indicates a shared table. Other values indicate the specified database and shared table. **rel_id** indicates the ID of the specified table, including all system catalogs that have system caches. The default value **NULL** or **–1** indicates all tables. Other values indicate the specified

table. If the database does not exist, an error is reported. If the table does not exist, a null result is returned.

Return type: tuple

Example:

```
-- Obtain the OID of a specific database through pg_database, for example, running SELECT oid, * FROM pg_database;.
-- In the returned tuple, find the value of the OID column based on the database name column, and then run the following query command. In the example, the obtained OID is 16574.
gaussdb=# SELECT * FROM gs_gsc_catalog_detail(16574, 1260);
 database_id | database_name | rel_id | rel_name  | cache_id |  self  |  ctid  | infomask | infomask2 | hash_value | refcount
-------------+---------------+--------+-----------+----------+--------+--------+----------+-----------+------------+----------
           0 |               |   1260 | pg_authid |       10 | (0, 9) | (0, 9) |    10507 |        26 |  531311568 |
10
           0 |               |   1260 | pg_authid |       11 | (0, 4) | (0, 4) |     2313 |        26 |  365368336 |        1
           0 |               |   1260 | pg_authid |       11 | (0, 9) | (0, 9) |    10507 |        26 | 3911517328 |
10
           0 |               |   1260 | pg_authid |       11 | (0, 7) | (0, 7) |     2313 |        26 | 1317799983 |
1
           0 |               |   1260 | pg_authid |       11 | (0, 5) | (0, 5) |     2313 |        26 | 3664347448 |
1
           0 |               |   1260 | pg_authid |       11 | (0, 1) | (0, 1) |     2313 |        26 |  276477273 |        1
           0 |               |   1260 | pg_authid |       11 | (0, 3) | (0, 3) |     2313 |        26 | 2465837659 |
1
           0 |               |   1260 | pg_authid |       11 | (0, 8) | (0, 8) |     2313 |        26 | 3205288035 |
1
           0 |               |   1260 | pg_authid |       11 | (0, 6) | (0, 6) |     2313 |        26 |  131811687 |        1
           0 |               |   1260 | pg_authid |       11 | (0, 2) | (0, 2) |     2313 |        26 | 1226484587 |
1
(10 rows)
```

- gs_gsc_clean(database_id default NULL)

  Description: Clears the global system cache. Note that data in use will not be cleared. The user who calls this function must have the SYSADMIN permission.

  Parameter: Specifies the database whose global system cache needs to be cleared. The default value **NULL** or value **–1** indicates that the global system cache of all databases is forcibly cleared. The value **0** indicates that the global system cache of only the shared table is cleared. Other values indicate that the global system cache of a specified database and a specified shared table is cleared. If **database_id** does not exist, an error is reported.

  Return type: Boolean

  Example:

  ```
  gaussdb=# SELECT * FROM gs_gsc_clean();
  gs_gsc_clean
  --------------
  t
  (1 row)
  ```

- gs_gsc_dbstat_info(database_id default NULL)

  Description: Obtains GSC memory statistics on the local node, including cache query, hit, loading, expiration, and occupied space information of tuples, relations, and partitions, database-level eviction information, thread reference information, and memory usage information. This function can be used to locate performance problems. For example, if the value of the hits or searches array is far less than 1, the value of **global_syscache_threshold** may be too small. As a result, the query hit ratio decreases. The user who calls this function must have the SYSADMIN permission.

Parameter: Specifies the global system cache statistics of the database to be queried. Value **NULL** or **–1** indicates that all databases are queried. Value **0** indicates that only the shared table is queried. Other values indicate that a specified database and a specified shared table are queried. If an invalid value is entered, an error is reported, indicating that **database_id** does not exist.

Return type: tuple

Example:

```
gaussdb=# SELECT * FROM gs_gsc_dbstat_info();
database_id | database_name | tup_searches | tup_hits | tup_miss | tup_count | tup_dead | tup_memory
| rel_searches | rel_hits | rel_mis
s | rel_count | rel_dead | rel_memory | part_searches | part_hits | part_miss | part_count | part_dead |
part_memory | total_memory | swa
pout_count | refcount
-------------+---------------+--------------+----------+----------+-----------+----------+------------
+--------------+----------+--------
--+-----------+----------+------------+---------------+-----------+-----------+------------+-----------
+------------+--------------+----
-----------+----------
          0 |               |          300 |      235 |       31 |        22 |        2 |       9752 |          598 |      108
|       1
8 |        18 |        0 |      77720 |            0 |         0 |         0 |         0 |         0 |          0 |       752912 |
         0 |         0
      16574 | testdb      |         3368 |     2289 |      329 |       273 |        0 |      92593 |         1113 |
524 |        4
8 |        48 |        0 |     340456 |            0 |         0 |         0 |         0 |         0 |          0 |      4124792 |
         0 |        10
(2 rows)
```

7.6.41 Data Damage Detection and Repair Functions

The restrictions for repairing file or page of the primary and standby nodes are as follows:

File Type	File/ Page	Primary/ Standby	Detection and Repair
Ordinary row-store tables (including Astore and Ustore) and segment-page tables (excluding indexes and compressed tables)	File and page	Primary	Manual detection and repair.
Undo files (excluding undo meta)	Page	Primary	Manual detection and repair (excluding analyse verify).
init fork file for unlogged tables	File	Primary	Manual detection and repair.

Ordinary row-store tables (including Astore and Ustore, but excluding the compressed table), indexes (including B-tree and UB-tree), and undo files (excluding undo meta. Undo files support only CRC errors verification.)	Page	Stand by	Automatic detection and repair during replay.

CAUTION

The first standby node and cascaded standby node of the standby cluster can be repaired.

A critical section is used in the database to ensure the consistency of access to key resources. Errors cannot be thrown in the critical section. If a damaged page is accessed in the critical area, PANIC is triggered. Repair detection and automatic repair are not supported. Typical operations that access the critical area during execution include database write operations such as INSERT, DELETE, UPDATE, and DROP.

- gs_verify_data_file(verify_segment bool)

  Description: Checks whether files in the current database of the current instance are lost. The verification only checks whether intermediate segments are lost in the main file of the data table. The default value is **false**, indicating that the segment-page table data file is not verified. If this parameter is set to **true**, only segment-page table files are verified. By default, only initial users, users with the sysadmin permission, and users with the O&M administrator permission in the O&M mode can view the information. Other users can view the information only after being granted with permissions.

  The returned result is as follows:

  – Non-segment-page table: **rel_oid** and **rel_name** indicate the table OID and table name of the corresponding file, and **miss_file_path** indicates the relative path of the lost file.

  – Segment-paged table: All tables are stored in the same file. Therefore, **rel_oid** and **rel_name** cannot display information about a specific table. For a segment-page table, if the first file is damaged, the subsequent files such as .1 and .2 are not checked. For example, if 3, 3.1, and 3.2 are damaged, only 3 damage can be detected. When the number of segment-page files is less than 5, the files that are not generated are also detected during function detection. For example, if there are only files 1 and 2, files 3, 4, and 5 are detected during segment-page file detection. In the following examples, the first is an example of checking a non-segment-page table, and the second is an example of checking a segment-page table.

  Parameter description:

  – verify_segment

Specifies the range of files to be checked. **false** indicates that non-segment-page tables are verified. **true** indicates that segment-page tables are verified.

The value can be **true** or **false** (default value).

Return type: record

Example (The abnormal line is displayed only when an exception is detected. Otherwise, no line is displayed.):

Verify a non-segment-page table.

```
gaussdb=# SELECT * FROM gs_verify_data_file();
node_name         | rel_oid |  rel_name    |  miss_file_path
------------------+---------+--------------+------------------
dn_6001_6002_6003 |   16554 |     test     | base/16552/24745
```

Verify a segment-page table.

```
gaussdb=# SELECT * FROM gs_verify_data_file(true);
     node_name     | rel_oid | rel_name | miss_file_path
-------------------+---------+----------+----------------
 dn_6001_6002_6003 |       0 | none     | base/16573/2
```

- gs_repair_file(tableoid Oid, path text, timeout int)

  Description: Repairs the file based on the input parameters. Only the primary DN with normal primary/standby connection is supported. Only the main file of the data table can be repaired. The parameters are set based on the OID and path returned by the gs_verify_data_file function. The value of table OID for a segment-page table ranges from 0 to 4294967295. (The internal verification determines whether a file is a segment-page table file based on the file path. The table OID is not used for a segment-page table file.) If the repair is successful, **true** is returned. If the repair fails, the failure cause is displayed. By default, only initial users, users with the sysadmin permission, and users with the O&M administrator permission in the O&M mode on the primary DN can view the information. Other users can view the information only after being granted with permissions.

  CAUTION

  1. If a file on a DN is damaged, a verification error at the PANIC level is reported when the DN is promoted to primary. The DN cannot be promoted to primary, which is normal.
  2. If a file exists but its size is 0, the file will not be repaired. To repair the file, you need to delete the file whose size is 0 and then repair it.
  3. You can delete a file only after the file descriptor is automatically closed. You can manually restart the process or perform a primary/standby switchover.

  Parameter description:

  – tableoid

    OID of the table corresponding to the file to be repaired. Set this parameter based on the **rel_oid** column in the list returned by the **gs_verify_data_file** function.

    Value range: OID ranging from 0 to 4294967295. Note: A negative value will be forcibly converted to a non-negative integer.

  - path

    Path of the file to be repaired. Set this parameter based on the **miss_file_path** column in the list returned by the **gs_verify_data_file** function.

    Value range: a string

  - timeout

    Specifies the duration for waiting for the standby DN to replay. The repair file needs to wait for the standby DN to be put back to the corresponding location on the current primary DN. Set this parameter based on the replay duration of the standby DN.

    Value range: 60s to 3600s.

  Return type: Boolean

  Example (Set **tableoid** and **path** based on the output of gs_verify_data_file):

  Page-based storage:

```
gaussdb=# SELECT * FROM gs_repair_file(16554,'base/16552/24745',360);
gs_repair_file
----------------
t
```

  Segment-page storage:

```
gaussdb=# SELECT * FROM gs_repair_file(16554,'base/16552/2',360);
gs_repair_file
----------------
t
```

- local_bad_block_info()

  Description: Displays the page damage of the instance. You can read the page from the disk and record the page CRC failure. By default, only initial users, users with the sysadmin permission, users with the monitoring administrator attribute, users with the O&M administrator attribute in the O&M mode, and monitoring users can view the information. Other users can view the information only after being granted with permissions.

  In the preceding information, **file_path** indicates the relative path of the damaged file. The segment-page mode are supported. **block_num** indicates the number of the page where the file is damaged. The page number starts from 0. **check_time** indicates the time when the page damage is detected. **repair_time** indicates the time when the page is repaired.

  Return type: record

  Example (Related entries are displayed only when there are damaged records. Otherwise, no log is displayed.):

```
gaussdb=# SELECT * FROM local_bad_block_info();
node_name    | spc_node | db_node | rel_node| bucket_node | fork_num | block_num |    file_path     |
check_time            |   repair_time
-----------------+--------+---------+---------+--------------+----------+-----------+------------------
+--------------------------+-------------------------------
dn_6001_6002_6003|  1663 |  16552 |  24745 |        -1    |    0    | 0        | base/16552/24745 |
2022-01-13 20:19:08.385004+08 | 2022-01-13 20:19:08.407314+08
```

- local_clear_bad_block_info()

  Description: Deletes data of repaired pages from **local_bad_block_info**, that is, information whose **repair_time** is not empty. By default, only initial users, users with the sysadmin permission, users with the O&M administrator attribute in the O&M mode, and monitoring users can view the information.

Other users can view the information only after being granted with permissions.

Return type: Boolean

Example:

```
gaussdb=# SELECT * FROM local_clear_bad_block_info();
result
--------
t
```

- gs_verify_and_tryrepair_page (path text, blocknum oid, verify_mem bool, is_segment bool)

  Description: Verifies the page specified by the instance. By default, only the initial user, users with the sysadmin permission, and users with the O&M administrator attribute in O&M mode on the primary DN can view the table. Other users can view the table only after being granted with permissions.

  In the command output, **disk_page_res** indicates the verification result of the page on the disk, **mem_page_res** indicates the verification result of the page in the memory, and **is_repair** indicates whether the repair function is triggered during the verification. **t** indicates that the page is repaired, and **f** indicates that the page is not repaired.

  NOTE

  - If a page on a DN is damaged, a verification error at the PANIC level is reported when the DN is promoted to primary. The DN cannot be promoted to primary, which is normal. Damaged pages of hash bucket tables cannot be repaired.
  - The repair triggered by this function can only repair pages in the memory. The repair takes effect only after the memory pages are flushed to disks and the physical pages are repaired.

  Parameter description:

  – path

    Path of the damaged file. Set this parameter based on the **file_path** column in the **local_bad_block_info** file. To verify the undo pages of the Ustore table, enter the path of the undo pages to be verified.

    Value range: a string

  – blocknum

    Page number of the damaged file. Set this parameter based on the **block_num** column in the **local_bad_block_info** file. If you want to verify the undo pages of the Ustore table, enter the block number of the undo pages to be verified.

    Value range: OID ranging from 0 to 4294967295. Note: A negative value will be forcibly converted to a non-negative integer.

  – verify_mem

    Specifies whether to verify a specified page in the memory. If this parameter is set to **false**, only pages on the disk are verified. If this parameter is set to **true**, pages in the memory and on the disk are verified. If a page on the disk is damaged, the system verifies the basic information of the page in the memory and flushes the page to the disk to restore the page. If a page is not found in the memory during memory page verification, the page on the disk is read through the memory API. During this process, if the disk page is faulty, the remote read automatic repair function is triggered.

Value range: The value is of the Boolean type and can be **true** or **false**.

– is_segment

Determines whether the table is a segment-page table. Set this parameter based on the value of **bucket_node** in the **local_bad_block_info** file. If the value of **bucket_node** is **–1**, the table is not a segment-page table. In this case, set **is_segment** to **false**. If the value of **bucket_node** is not –1, set **is_segment** to **true**. Segment-page storages are supported.

Value range: The value is of the Boolean type and can be **true** or **false**.

Return type: record

Examples (Transfer parameters based on the output of local_bad_block_info. Otherwise, an error is reported.):

Page-based storage:

```
gaussdb=# SELECT * FROM gs_verify_and_tryrepair_page('base/16552/24745',0,false,false);
node_name        |       path       |  blocknum  |         disk_page_res         | mem_page_res | is_repair
-----------------+------------------+------------+-------------------------------+--------------+----------
dn_6001_6002_6003 | base/16552/24745 |     0      | page verification succeeded.|              | f
```

Segment-page storage:

```
gaussdb=# SELECT * FROM gs_verify_and_tryrepair_page('base/14365/1',4494,false,true);
node_name        |     path     | blocknum |         disk_page_res         | mem_page_res | is_repair
-----------------+--------------+----------+-------------------------------+--------------+-----------
dn_6001_6002_6003 | base/14365/1 |     4494 | page verification succeeded. |              | f
```

- gs_repair_page(path text, blocknum oid, is_segment bool, timeout int)

  Description: Restores the specified page of the instance. This function can be used only by the primary DN that is properly connected to the primary and standby DNs. If the page is successfully restored, **true** is returned. If an error occurs during the restoration, an error message is displayed. By default, only the initial user, users with the sysadmin permission, and users with the O&M administrator attribute in O&M mode on the primary DN can view the table. Other users can view the table only after being granted with permissions.

  Note: If a page on a DN is damaged, a verification error at the PANIC level is reported when the DN is promoted to primary. The DN cannot be promoted to primary, which is normal. Damaged pages of hash bucket tables cannot be repaired.

  Parameter description:

  – path

  Path of the damaged page. Set this parameter based on the **file_path** column in **local_bad_block_info** or the **path** column in the **gs_verify_and_tryrepair_page** function.

  Value range: a string

  – blocknum

  Number of the damaged page. Set this parameter based on the **block_num** column in **local_bad_block_info** or the **blocknum** column in the **gs_verify_and_tryrepair_page** function.

  Value range: OID ranging from 0 to 4294967295. Note: A negative value will be forcibly converted to a non-negative integer.

  – is_segment

  Determines whether the table is a segment-page table. The value of this parameter is determined by the value of **bucket_node** in

local_bad_block_info. If the value of **bucket_node** is **–1**, the table is not a segment-page table and **is_segment** is set to **false**. If the value of **bucket_node** is not –1, **is_segment** is set to true. Segment-page storages are supported.

Value range: The value is of a Boolean type and can be **true** or **false**.

– timeout

Duration of waiting for standby DN replay. The page to be repaired needs to wait for the standby DN to be played back to the location of the current primary DN. Set this parameter based on the replay duration of the standby DN.

Value range: 60s to 3600s.

Return type: Boolean

Examples (Transfer parameters based on the output of local_bad_block_info. Otherwise, an error is reported.):

Page-based storage:

```
gaussdb=# SELECT * FROM gs_repair_page('base/16552/24745',0,false,60);
result
--------
t
```

Segment-page storage:

```
gaussdb=# SELECT * FROM gs_repair_page('base/16552/1',4494,true,60);
result
--------
t
```

- gs_seg_verify_datafile(IN tablespace_name name, IN database_name name, IN file_id integer, IN bucketnode integer, IN start_block_id bigint default 0, IN end_block_id bigint default UINT32MAX)

  a. Description

  Description: Verifies segment-page files 1 to 5, checks whether the main fork has page damage, and records the verification result in the local_bad_block_info() function.

  Permission: Only users with the sysadmin attribute and users with the O&M administrator attribute in O&M mode can execute this function.

  b. Parameter description:

Name	Type	Description
tablespace_name	name	Tablespace name.
database_name	name	Database name.
file_id	integer	File name. The value ranges from 1 to 5.

Name	Type	Description
bucketnode	integer	Bucket node of the table. • **0** to **1023** indicate the bucket nodes of a hash bucket table. • **1024** indicates the bucket node of a segment-page ordinary table. Currently, only hash bucket tables and segment-page ordinary tables are supported.
start_block_id	bigint	Start value of the verification page range. The default value is **0**.
end_block_id	bigint	End value of the verification page range. The default value is **4294967295**.

c. Return value: None

Example:

```
gaussdb=# SELECT * FROM gs_seg_verify_datafile('seg_tblspc', 'postgres', 2, 1024);
WARNING:  page verification failed, calculated checksum 60994 but expected 11565, the block num is 4157
 gs_seg_verify_datafile
------------------------

(1 row)
```

- gs_edit_page_bypath(path text, blocknum int64, offset int, data text, data_size int, read_backup bool, storage_type text)

  Description: Transfers the path, block number, offset, target data to be modified, and length of the target table file, modifies the target data to the corresponding fields on the page, and returns the path of the modified file to be flushed to the disk. The **read_backup** column determines the file reading mode, and the **storage_type** column indicates the file storage mode (for example, segment-page storage). To prevent incorrect modification, this function does not directly modify the original page but modifies the copied page and flushes the modified page to the specified path. Only the system administrator or O&M administrator in O&M mode can execute this function.

  Return type: text

Table 7-159 gs_edit_page_bypath parameters

Category	Parameter	Type	Description
Input parameter	path	text	Physical file path of the file to be modified, which is related to the **read_backup** field. The value can be the relative path of the file in the database directory or the absolute path of files such as the backup file. If the target file does not exist or fails to be read, an error message is displayed. ● If **read_backup** is **false**, the path format is *tablespace name*|*database oid*|*table relfilenode* (*physical file name*). For example, **base/16603/16394**. ● If **read_backup** is **true**, **path** is a valid path. In this case, because other information about the input file cannot be obtained, you need to ensure that the input data is correct. In a page-based file, only U-page and UB-tree data pages can be edited and modified. In a segment-page file, Astore data pages can be edited and modified. Tables with tablespaces are not supported. Other information about the input file cannot be obtained. Therefore, you need to ensure that the input data type is correct.
Input parameter	blocknum	bigint	Block number of the page to be repaired. Value range: 0 to *MaxBlockNumber*. Reads the page corresponding to the specified physical or logical block number based on the **read_backup** field. If the specified block number is out of range, an error message is returned.
Input parameter	offset	int	In-page offset of the field to be modified. Value range: 0 to *BLCKSZ* (*BLCKSZ* is excluded.) If the specified value is less than **0** or greater than or equal to that of **BLCKSZ**, the system view is used to return the corresponding error information.
Input parameter	data	text	Type of the target value to be modified. Type: ● **'0x'**: hexadecimal. ● **'0b'**: binary. ● **'0s'**: character string. Others: If the value of the **data** parameter is not one of the preceding types, the value can only be a decimal numeric string.

Category	Parameter	Type	Description
Input parameter	data_size	int	Length of the written data, in bytes. Value range: 1 to 8. If the specified write length is less than 1 byte or greater than 8 bytes, or the sum of offset and data_size is greater than the value of **BLCKSZ**, the system view is used to return the corresponding error information.
Input parameter	read_backup	bool	Specifies whether to read pages from the backup directory. If this parameter is set to **false**, the target page is read based on the logical block number. Otherwise, the page is read based on the physical block number.
Input parameter	storage_type	text	Specifies the file storage mode. This parameter is optional. ● **'page'**: page mode. ● **'segment'**: segment-page mode.
Output parameter	output_msg	text	If the modification is successful, the absolute path of the modified file is returned. The modified file is stored in the **gs_log/dump** directory. If the modification fails, a failure message is returned.

In the example, transfer parameters based on the parameter description and use the actual physical path.

Example 1: Overwrite the data whose value is **0X1FFF** at the offset of 16 bytes on page 0 in the **base/15808/25075** table.

```
gaussdb=# SELECT gs_edit_page_bypath('base/15808/25075',0,16,'0x1FFF', 2, false, 'page');
                        gs_edit_page_bypath
-------------------------------------------------------------------------
/gs_log_dir/dump/1663_15808_25075_0.editpage

(1 rows)
```

Example 2: If the input parameter does not comply with the specifications, an error message is returned.

```
gaussdb=# SELECT gs_edit_page_bypath('base/15808/25075', 0,16,'@1231!', 8, false, 'page');
            gs_edit_page_bypath
-------------------------------------------
 Error: the parameter 'data' decode failed.
(1 row)
```

Example 3: When the data to be written is the same as the original value, an alarm is returned.

```
gaussdb=# SELECT gs_edit_page_bypath('/gs_log_dir/dump/1663_15808_25075_0.editpage',
0,16,'0x1FFF', 2, true, 'page');
                    gs_edit_page_bypath
-----------------------------------------------------------
 Warning: source buffer is consistent with target buffer.
(1 row)
```


- gs_repair_page_bypath(src_path text, src_blkno int64, dest_path text, dest_blkno int64, storage_type text)

  Description: Transfers the path and page number of the source file, and writes the page to the specified page number of the target file. You can repair pages of the standby node through the primary node or repair the pages of the primary node through the standby node. In addition, you can initialize bad blocks in this view.

  a. The target page is overwritten and synchronized to the standby node. The page-based modification object supports the U-heap and UB-tree pages. The Undo Record page, Undo Slot page, compressed table, and Astore page will be supported later. The segment-page modification object supports the Astore pages. System catalog files and data sections cannot be modified.
  b. With this function, you can overwrite target pages during the write operation. Before overwriting, the target page is backed up and flushed to a specified directory. The backup page can be rewritten back to the target page. If an ordinary table is modified on the primary node, a new WAL is generated and synchronized to the standby node. If an ordinary table is modified on the standby node, no WAL is recorded.
  c. The repair view applies only to the primary node in a centralized or distributed system or the standby node when the read function is enabled on the standby node. Only the system administrator or O&M administrator in O&M mode can use this function. All modifications will be recorded in database logs. In addition, you are advised to enable the audit logging function of system functions before using this function to record audit information.
  d. When the repair view is called on the standby node, the standby node can be repaired only by reading pages from the primary node.
  e. The LSNs of the source and target pages must be the same. Otherwise, the repair fails.

  Return type: text

  CAUTION

  Invoking this system function is a high-risk operation. Exercise caution when performing this operation.

Category	Parameter	Type	Description
Input parameter	src_path	text	Path of the source file. The following types of paths are supported: • Data files and index files: **gs_log/dump/1663_15808_25075_0.editpage**. • **src_path** is set to **'standby'** on the primary node. That is, pages are read from the standby node to repair the primary node. • **src_path** is set to **'primary'** on the standby node. That is, pages are read from the primary node to repair the standby node. On the standby node, **src_path** can only be set to **'primary'** and the read function must be enabled on the standby node before the repair. • **src_path** is set to **init_block** on the primary node to allow skipping bad blocks in extreme scenarios.
Input parameter	src_blkno	bigint	Physical block number of the source page. Value range: 0 to *MaxBlockNumber*.
Input parameter	dest_path	text	Relative path of the target file. For example, **base/15808/25075**.
Input parameter	dest_blkno	bigint	Logical block number of the target page. Value range: 0 to *MaxBlockNumber*.
Input parameter	storage_type	text	Specifies the target file storage mode. This parameter is optional. • **'page'**: page mode. • **'segment'**: segment-page mode.
Output parameter	output_msg	text	If the write overwrite operation is successful, the backup path of the target page is returned. If the write overwrite operation fails, an error message is returned. The format of the flushed file name is *relfilepath_blocknum_timestamp.repairpage*.

Transfer parameters based on the preceding table and ensure that the physical file exists. If the input parameter is abnormal or the restoration fails, an error is reported.

Example 1: Enter a file in a specified path to overwrite the target file.

```
gaussdb=# SELECT * FROM gs_repair_page_bypath('gs_log/dump/1663_15991_16767_0.editpage', 0,
'base/15991/16767', 0, 'page');
                                        output_msg
-------------------------------------------------------------------------------------------------
 /gs_log_dir/dump/1663_15991_16767_0_738039702421788.repairpage
(1 row)
```

Example 2: Read pages from the standby node to repair the primary node.

```
gaussdb=# SELECT * FROM gs_repair_page_bypath('standby', 0, 'base/15990/16768', 0, 'page');
                                        output_msg
-------------------------------------------------------------------------------------------------
/gs_log_dir/dump/1663_15990_16768_0_738040397197907.repairpage
(1 row)
```

Example 3: Read pages from the primary node to repair the standby node.

```
gaussdb=# SELECT * FROM gs_repair_page_bypath('primary', 0, 'base/15990/16768', 0, 'page');
                                        output_msg
-------------------------------------------------------------------------------------------------
 /gs_log_dir/dump/1663_15990_16768_0_738040506157799.repairpage
(1 row)
```

Example 4:

```
gaussdb=# SELECT * FROM gs_repair_page_bypath('init_block', 0, 'base/15990/16768', 0, 'page');
                                        output_msg
-------------------------------------------------------------------------------------------------
/gs_log/dump/1663_15990_16768_0_738040768010281.repairpage
(1 row)
```

- gs_repair_undo_byzone(zone_id int)

  Description: Transfers the ID of the undo zone to be repaired, repairs the metadata of the target undo zone, and returns the repair result details. If the undo zone is not repaired, no information is output.

  Return type: record

  Note: Currently, the function can be called only on the primary node. After the repair is successful, the repair will be synchronized to the standby node by recording Xlogs. The caller must be a system administrator or an O&M administrator in O&M mode. You are advised to enable the audit logging function before using the function to record audit information.

  CAUTION

  Invoking this system function is a high-risk operation. Exercise caution when performing this operation.

Table 7-160 gs_repair_undo_byzone parameters

Category	Parameter	Type	Description
Input parameter	zone_id	int	Undo zone ID: • **–1**: repairs the metadata of all undo zones. • **0** to **1048575**: repairs the metadata of the undo zone corresponding to the zone ID.
Output parameter	zone_id	int	Undo zone ID.
Output parameter	repair_detail	text	Repair result of the undo zone metadata corresponding to the zone ID. If the repair is successful, "rebuild undo meta succeed." is displayed. If the repair fails, "rebuild undo meta failed." as well as the failure cause is displayed.

Example 1: If the undo zone meta information corresponding to the entered **zone_id** is not damaged, no output is expected.

```
gaussdb=# SELECT * FROM gs_repair_undo_byzone(4);
 zone_id | repair_detail
---------+---------------
(0 rows)
```

Example 2: If the undo zone metadata corresponding to the entered **zone_id** is successfully restored, the system displays a message indicating that the restoration is successful.

```
gaussdb=# SELECT * FROM gs_repair_undo_byzone(78);
 zone_id | repair_detail
---------+---------------
      78 | rebuild undo meta succeed.
(1 row)
```

Example 3: If the undo zone metadata corresponding to the entered zone ID fails to be repaired, the detailed information about the repair failure is displayed.

```
gaussdb=# SELECT * FROM gs_repair_undo_byzone(0);
 zone_id | repair_detail
---------+---------------
       0 | rebuild undo meta failed. try lock undo zone_id failed.
(1 row)
```

NOTE

If the undo zone to be repaired is damaged and the zone ID is occupied by another active thread, the active thread that occupies the zone ID automatically ends when the repair function is called to forcibly repair the damaged undo zone metadata.

- gs_verify_urq(index_oid oid, partindex_oid oid, blocknum bigint, queue_type text)

  Description: Verifies the correctness of the index recycling queue (potential queue/available queue/single page).

  Parameter description: See **Table 7-161**.

Return type: record

Table 7-161 gs_verify_urq parameters

Category	Parameter	Type	Description
Input parameter	index_oid	oid	UB-tree index OID. • Common index: index OID. • Global index: GPI OID. • Local index: OID of the primary index.
Input parameter	partindex_oid	oid	UB-Tree partitioned index OID: • Common index: 0. • Global index: 0. • Local index: OID of the partitioned index (primary or secondary).
Input parameter	blocknum	bigint	Specifies the page number: • If the queue type is single page, the correctness of all tuples of **blocknum** on a single page is verified. The value range is [0,*Queue file size*/8192). • If the queue is empty or free, **blocknum** is an invalid value.
Input parameter	queue_type	text	Specifies the queue type: • **empty queue**: potential queue • **free queue**: available queue • **single page**: single-page queue
Output parameter	error_code	text	Error code
Output parameter	detail	text	Detailed error information and other key information.

When using the example, transfer parameters based on the parameter description and use the actual OID and **blocknum**. Otherwise, an error is reported.

Example 1:

```
gaussdb=# SELECT * FROM gs_verify_urq(16387, 0, 1, 'free queue');
 error_code | detail
------------+--------
(0 rows)
```


Example 2:

```
gaussdb=# SELECT * FROM gs_verify_urq(16387, 0, 1, 'empty queue');
     error_code       |
detail
----------------------
+-----------------------------------------------------------------------------------------------------------------
 VERIFY_URQ_PAGE_ERROR | invalid urq meta: oid 16387, blkno 1, head_blkno = 1, tail_blkno = 3,
nblocks_upper = 4294967295, nblocks_lower = 1; urq_blocks = 6, index_blocks = 12
(1 row)
```

NOTE

Currently, this API supports only Ustore index tables. If the verification of the index recycling queue is normal, the view does not display the error code and error details. Otherwise, the view displays the error code and error details. The error codes include "VERIFY_URQ_PAGE_ERROR", "VERIFY_URQ_LINK_ERROR", "VERIFY_URQ_HEAD_MISSED_ERROR", and "VERIFY_URQ_TAIL_MISSED_ERROR". If any of the preceding error codes is displayed, contact Huawei engineers to locate the fault.

- gs_urq_dump_stat(index_oid oid, partindex_oid oid)

  Description: Queries information about a specified index recycling queue.

  In the return result, **recentGlobalDataXmin** and **globalFrozenXid** are two oldestxmins used by the recycling queue to determine whether the index page can be recycled, **next_xid** is the XID of the next latest transaction, **urq_blocks** indicates the total number of pages in the recycling queue and information about valid pages in the free queue (available queue) and empty queue (potential queue). For PCR indexes, the output recentGlobalDataXmin is replaced by globalRecycleXid that affects the recycling of the PCR index pages.

  Parameters: For details, see **Table 7-162**.

  Table 7-162 gs_urq_dump_stat parameters

Category	Parameter	Type	Description
Input parameter	index_oid	oid	UB-tree index OID. • Common index: index OID. • Global index: GPI OID. • Local index: OID of the primary index.
Input parameter	partindex_oid	oid	UB-Tree partitioned index OID: • Common index: 0. • Global index: 0. • Local index: OID of the partitioned index (primary or secondary).
Output parameter	result	text	Detailed statistics about the index recycling queue.

When using the example, transfer parameters based on the parameter description and use the actual OID. Otherwise, an error is reported.

Example (RCR):

```
gaussdb=# SELECT * FROM gs_urq_dump_stat(16387, 0);
                                                    result
------------------------------------------------------------------------------------------------------------------------------
----------
 urq stat info: recentGlobalDataXmin = 213156, globalFrozenXid = 213156, next_xid = 214157,
urq_blocks = 6,                    +
   free  queue:    head page blkno = 0 min_xid = 211187 max_xid = 214157, tail page blkno = 0
min_xid = 211187 max_xid = 214157,+
              middle page min_xid = 1152921504606846975 max_xid = 0, valid_pages = 1, valid_items =
6, can_use_item = 3        +
   empty queue:    head page blkno = 1 min_xid = 212160 max_xid = 213160, tail page blkno = 3
min_xid = 213162 max_xid = 214156,+
              middle page min_xid = 1152921504606846975 max_xid = 0, valid_pages = 2, valid_items =
999, can_use_item = 498   +

(1 row)
```

Example (PCR):

```
gaussdb=# SELECT gs_urq_dump_stat(17260,0);
                                                       gs_urq_dump_stat
------------------------------------------------------------------------------------------------------------------------------
---------------------------
 urq stat info: globalRecycleXid = 22113, globalFrozenXid = 22107, next_xid = 22116, urq_blocks =
6,                                        +
   free  queue:    head page blkno = 0 min_xid = 1152921504606846975 max_xid = 0, tail page blkno
= 0 min_xid = 1152921504606846975 max_xid = 0,+
              middle page min_xid = 1152921504606846975 max_xid = 0, valid_pages = 1, valid_items =
0, can_use_item = 0                      +
   empty queue:    head page blkno = 1 min_xid = 1152921504606846975 max_xid = 0, tail page blkno
= 1 min_xid = 1152921504606846975 max_xid = 0,+
              middle page min_xid = 1152921504606846975 max_xid = 0, valid_pages = 1, valid_items =
0, can_use_item = 0                      +

(1 row)
```

NOTE

Currently, this API supports only Ustore index tables.

- gs_repair_urq(index_oid oid, partindex_oid oid)

  Description: Repairs (with loss) index recycling queues (potential and available queues). The recycling queue file of the current index is deleted and an empty recycling queue file is created. If the repair is successful, "reinitial the recycle queue of index relation successfully" is displayed.

  Parameter description: See **Table 7-163**.

  Note: The current function can be called only on the primary node.

Table 7-163 gs_repair_urq parameters

Category	Parameter	Type	Description
Input parameter	index_oid	oid	UB-tree index OID. ● Common index: index OID. ● Global index: GPI OID. ● Local index: OID of the primary index.

Category	Parameter	Type	Description
Input parameter	partindex_oid	oid	UB-Tree partitioned index OID: • Common index: 0. • Global index: 0. • Local index: OID of the partitioned index (primary or secondary).
Output parameter	result	text	When the rebuilding is successful, "reinitial the recycle queue of index relation successfully" is displayed. Otherwise, the rebuilding fails.

When using the example, transfer parameters based on the parameter description and use the actual OID. Otherwise, an error is reported.

Example:

```
gaussdb=# SELECT * FROM gs_repair_urq(16387, 0);
                          result
-----------------------------------------------------------
 reinitial the recycle queue of index relation successfully.
(1 row)
```

NOTE

Currently, this API supports only Ustore index tables.

- gs_get_standby_bad_block_info()

  Description: Displays the pages that have been detected on the standby node but have not been repaired. By default, only initial users, users with the sysadmin permission, users with the O&M administrator permission in the O&M mode, and users with the monitor administrator permission on the standby DN can view the information. Other users can view the information only after being granted with permissions. There are four return values in the **invalid_type** column: **NOT_PRESENT** (the page does not exist), **NOT_INITIALIZED** (the page initialization fails), **LSN_CHECK_ERROR** (the LSN check fails), and **CRC_CHECK_ERROR** (the CRC check fails).

  Return type: record

  Example (If no page is detected but not repaired, no line is displayed.)

```
gaussdb=# SELECT * FROM gs_get_standby_bad_block_info();
 spc_node | db_node | rel_node | bucket_node | fork_num | block_num |  invalid_type   |
master_page_lsn
----------+---------+----------+-------------+----------+-----------+-----------------+-----------------
   1663   |   16552 |   24745  |      -1     |    0     |     0     | CRC_CHECK_ERROR | 0/B2009E8
(1 rows)
```

7.6.42 Functions of the XML Type

The following functions are inherited from open source PostgreSQL 9.2.

NOTE

In all the following XML functions, when the GUC parameter **xmloption** is set to **content**, the value of **encoding** in the XML declaration can be **ZHS16GBK**. When the GUC parameter **xmloption** is set to **document**, the value of **encoding** in the XML declaration cannot be **ZHS16GBK**. If **encoding** is set to **ZHS16GBK**, an error is reported.

- xmlparse ({ DOCUMENT | CONTENT } value [wellformed])

  Description: Generates XML values from character data.

  Parameter: data of the TEXT type

  Return type: XML

  Example:

```
gaussdb=# SELECT XMLPARSE (DOCUMENT '<?xml version="1.0"?><book><title>Manual</
title><chapter>...</chapter></book>');
                  xmlparse
-----------------------------------------------------------
 <book><title>Manual</title><chapter>...</chapter></book>
(1 row)
gaussdb=# SELECT XMLPARSE (CONTENT 'abc<foo>bar</foo><bar>foo</bar>');
         xmlparse
---------------------------------
 abc<foo>bar</foo><bar>foo</bar>
(1 row)
gaussdb=# SELECT XMLPARSE (CONTENT 'abc<foo>bar</foo' wellformed);
    xmlparse
------------------
 abc<foo>bar</foo
(1 row)
```

- xmlserialize({ DOCUMENT | CONTENT } value AS type)

  Description: Generates a string from an XML file.

  Parameter: The type can be character, character varying, text, or any variant of them.

  Return type: XML

  Example:

```
gaussdb=# SELECT XMLSERIALIZE(CONTENT 'good' AS CHAR(10));
 xmlserialize
--------------
 good
(1 row)
gaussdb=# SELECT xmlserialize(DOCUMENT '<head>bad</head>' as text);
  xmlserialize
------------------
 <head>bad</head>
(1 row)
```

  NOTE

  If a string value is converted to XML without using the xmlparse or xmlserialize function, the **XML OPTION** session parameter determines the value, **DOCUMENT** or **CONTENT**. The **XML OPTION** session parameter can be set by the standard command.

```
SET XML OPTION { DOCUMENT | CONTENT };
```

  Or use similar syntax to set this parameter.

```
SET xmloption TO { DOCUMENT | CONTENT };
```

- xmlcomment(text)

  Description: Creates an XML value that contains an XML comment with the specified text as the content. The text does not contain the "--" character and does not end with a "-" character. Besides, the text should meet the format

requirements of XML comments. If the parameter is empty, the result is also empty.

Parameter: data of the TEXT type

Return type: XML

Example:

```
gaussdb=# SELECT xmlcomment('hello');
  xmlcomment
--------------
 <!--hello-->
```

- xmlconcat(xml[, ...])

  Description: Concatenates a list of single XML values into a single value that contains an XML content fragment. Null values are ignored, and the result is null only when all parameters are null. In database A-compatible mode, you can set **a_format_version** to **10c** and **a_format_dev_version** to **s2** to check whether the input segment is well-formed XML text.

  Parameter: data of the XML type

  Return type: XML

  Note: Example 2 is a syntax example compatible with the A database.

  Example 1:

```
gaussdb=# set xmloption=content;
SET
gaussdb=# select XMLCONCAT(('<?xml version="1.0" encoding="GB2312" standalone="no"?
><bar>foo</bar>'),('<?xml version="1.0" encoding="GB2312" standalone="no" ?><bar>foo</bar>')) ;
                          xmlconcat
----------------------------------------------------------------------
<?xml version="1.0" standalone="no"?><bar>foo</bar><bar>foo</bar>
(1 row)
gaussdb=# select XMLCONCAT('abc>');
 xmlconcat
-----------
 abc>
(1 row)
```

  Example 2:

```
gaussdb=# SET a_format_version='10c';
SET
gaussdb=# SET a_format_dev_version=s2;
SET
gaussdb=# SET xmloption=content;
SET
gaussdb=# SELECT XMLCONCAT(('<?xml version="1.0" encoding="GB2312" standalone="no"?
><bar>foo</bar>'),('<?xml version="1.0" encoding="GB2312" standalone="no" ?><bar>foo</bar>')) ;
                          xmlconcat
----------------------------------------------------------------------
<?xml version="1.0" standalone="no"?><bar>foo</bar><bar>foo</bar>
(1 row)
gaussdb=# SELECT XMLCONCAT('abc>');
ERROR:  invalid XML document
DETAIL:  line 1: Start tag expected, '<' not found
abc>
^
CONTEXT:  referenced column: xmlconcat
```

  NOTE

  In A-compatible database, if **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s2**, and the **encoding** attribute in the XML declaration is set to **ZHS16GBK**, the XMLCONCAT function reports an error.

- xmlelement([ENTITYESCAPING | NOENTITYESCAPING] { [NAME] element_name | EVALNAME element_name } [,

xmlattributes([ENTITYESCAPING | NOENTITYESCAPING] value [[AS] attname | AS EVALNAME attname] [, ...])] [, content [[AS] alias] [, ...]])

Description: Generates an XML element with the given name, attribute, and content.

Return type: XML

Example:

```
gaussdb=# SELECT xmlelement(name foo);
 xmlelement
------------
 <foo/>

In compatible mode A:
gaussdb=# set a_format_version='10c';
SET
gaussdb=# set a_format_dev_version=s2;
SET
1. If the keyword ENTITYESCAPING is not set in XMLElement by default or is set, the reserved characters in the content of XMLElement are escaped.
gaussdb=# SELECT xmlelement("entityescaping<>", 'a$><&"b');
                      xmlelement
-------------------------------------------------------------
 <entityescaping<>>a$&gt;&lt;&amp;&quot;b</entityescaping<>>
(1 row)

gaussdb=# SELECT xmlelement(entityescaping "entityescaping<>", 'a$><&"b');
                      xmlelement
-------------------------------------------------------------
 <entityescaping<>>a$&gt;&lt;&amp;&quot;b</entityescaping<>>
(1 row)

2. When the keyword NOENTITYESCAPING is set in XMLElement, the reserved characters in the content of XMLElement will not be escaped.
gaussdb=# SELECT xmlelement(noentityescaping "entityescaping<>", 'a$><&"b');
                xmlelement
-----------------------------------------------
 <entityescaping<>>a$><&"b</entityescaping<>>
(1 row)

3. When [AS] alias is used to declare an alias for the content in XMLElement, the content value type must be XML.
gaussdb=# SELECT xmlelement("entityescaping<>", '<abc/>' b);
ERROR:  argument of XMLELEMENT must be type xml, not type unknown
LINE 1: SELECT xmlelement("entityescaping<>", '<abc/>' b);
                                              ^
CONTEXT:  referenced column: xmlelement

gaussdb=# SELECT xmlelement("entityescaping<>", '<abc/>' as b);
ERROR:  argument of XMLELEMENT must be type xml, not type unknown
LINE 1: SELECT xmlelement("entityescaping<>", '<abc/>' as b);
                                              ^
CONTEXT:  referenced column: xmlelement

gaussdb=# SELECT xmlelement("entityescaping<>", xml('<abc/>') b);
               xmlelement
-----------------------------------------------
 <entityescaping<>><abc/></entityescaping<>>
(1 row)

gaussdb=# SELECT xmlelement("entityescaping<>", xml('<abc/>') as b);
               xmlelement
-----------------------------------------------
 <entityescaping<>><abc/></entityescaping<>>
(1 row)

4. If the keyword ENTITYESCAPING is not set in XMLAttributes by default or is set, the reserved
```



```
characters in XMLAttributes are escaped.
gaussdb=# SELECT xmlelement("entityescaping<>", xmlattributes('entityescaping<>'
"entityescaping<>"));
                       xmlelement
-------------------------------------------------------------
 <entityescaping<> entityescaping<>="entityescaping&lt;&gt;"/>
(1 row)

gaussdb=# SELECT xmlelement(name "entityescaping<>", xmlattributes(entityescaping
'entityescaping<>' "entityescaping<>"));
                       xmlelement
-------------------------------------------------------------
 <entityescaping<> entityescaping<>="entityescaping&lt;&gt;"/>
(1 row)

5. When the NOENTITYESCAPING keyword is set in XMLAttributes, the reserved characters in
XMLAttributes will not be escaped.
gaussdb=# SELECT xmlelement("entityescaping<>", xmlattributes(noentityescaping 'entityescaping<>'
"entityescaping<>"));
                     xmlelement
-----------------------------------------------------------
 <entityescaping<> entityescaping<>="entityescaping<>"/>
(1 row)
```

NOTE

1. For **xmlelement** and **xmlattributes**, when the value of **name** is **NULL**, the database behavior is different from that of the A database. When the **name** field of **xmlelement** is set to **NULL**, the name information is empty and the attribute information is not displayed. When the **name** field of **xmlattributes** is set to **NULL**, the attribute information is not displayed.
2. After the following two parameters are set, the content escape rule of **xmlelement** is A-compatible. If the two parameters are not set, the content escape rule of **xmlelement** is PG-compatible.

```
set a_format_version='10c';
set a_format_dev_version=s2;
```

- xmlforest(content [AS name] [, ...])

  Description: Generates an XML sequence of elements using the given name and content.

  Return type: XML

  Example:

```
gaussdb=# SELECT xmlforest('abc' AS foo, 123 AS bar);
          xmlforest
------------------------------
 <foo>abc</foo><bar>123</bar>
```

- xmlpi(name target [, content])

  Description: Creates an XML processing instruction. If the content is not empty, the content cannot contain character collations.

  Return type: XML

  Example:

```
gaussdb=# SELECT xmlpi(name php, 'echo "hello world";');
            xmlpi
------------------------------
 <?php echo "hello world";?>
```

- xmlroot(xml, version text | no value [, standalone yes|no|no value])

  Description: Modifies the attributes of the root node of an XML value. If a version is specified, it replaces the value in the version declaration of the root node. If a standalone property is specified, it replaces the value of standalone declaration in the root node.

Example:

```
gaussdb=# SELECT xmlroot('<?xml version="1.1"?><content>abc</content>',version '1.0', standalone
yes);
                        xmlroot
---------------------------------------------------------------
 <?xml version="1.0" standalone="yes"?><content>abc</content>
(1 row)
```

- xmlagg(xml [order_by_clause])

  Description: Aggregates the input values called by the aggregate function and supports cross-row concatenation. For details about order_by_clause, see **SELECT**. In database A-compatible mode, you can set **a_format_version** to **10c** and **a_format_dev_version** to **s2**. The **xmloption** parameter of the database is set to **content** by default. When **xmloption** is set to **document**, linefeed characters are used to concatenate multiple XML lines. If the encoding attribute value in the XML declaration is not the default UTF-8, the aggregation result contains the XML declaration.

  Parameter: XML

  Return type: XML

  Note: Example 2 is a syntax example compatible with the A database.

  Example 1:

```
gaussdb=# CREATE TABLE xmltest (
           id int,
           data xml
          );
gaussdb=# INSERT INTO xmltest VALUES (1, '<value>one</value>');
INSERT 0 1
gaussdb=# INSERT INTO xmltest VALUES (2, '<value>two</value>');
INSERT 0 1
gaussdb=# SELECT xmlagg(data) FROM xmltest;
              xmlagg
--------------------------------------
 <value>one</value><value>two</value>
(1 row)
```

  Example 2:

```
gaussdb=# set xmloption=document;
SET
gaussdb=# SELECT xmlagg(data) FROM xmltest;
      xmlagg
--------------------
 <value>one</value>+
 <value>two</value>
(1 row)
gaussdb=# DELETE FROM XMLTEST;
DELETE 2
gaussdb=# INSERT INTO xmltest VALUES (1, '<?xml version="1.0" encoding="GBK"?><value>one</
value>');
INSERT 0 1
gaussdb=# INSERT INTO xmltest VALUES (2, '<?xml version="1.0" encoding="GBK"?><value>two</
value>');
INSERT 0 1
gaussdb=# SELECT xmlagg(data) FROM xmltest;
                        xmlagg
-----------------------------------------------------------
 <?xml version="1.0" encoding="GBK"?><value>one</value>+
 <value>two</value>
(1 row)
gaussdb=# SELECT xmlagg(data order by id desc) FROM xmltest;
                        xmlagg
-----------------------------------------------------------
 <?xml version="1.0" encoding="GBK"?><value>two</value>+
 <value>one</value>
(1 row)
```



```
gaussdb=# DROP TABLE xmltest;
```

- xmlexists(text passing [BY REF] xml [BY REF])

  Description: Evaluates an XPath 1.0 expression (the first parameter) with the passed XML value as its context item. If the evaluation result generates an empty set of nodes, the function returns **false**. If any other value is generated, the function returns **true**. If the value of any parameter is null, the function returns **Null**. The non-null value passed as a context item must be an XML document, not a content fragment or any non-XML value.

  Parameter: XML

  Return type: Boolean.

  Example:

```
gaussdb=# SELECT xmlexists('//town[text() = ''Toronto'']' PASSING BY REF '<towns><town>Toronto</
town><town>Ottawa</town></towns>');
 xmlexists
------------
 t
(1 row)
```

- xml_is_well_formed(text)

  Description: Checks whether the text is in the correct XML format and returns a Boolean value.

  Parameter: text

  Return type: Boolean.

  Example:

```
gaussdb=# SELECT xml_is_well_formed('<>');
 xml_is_well_formed
--------------------
 f
(1 row)
```

- xml_is_well_formed_document(text)

  Description: Checks whether the text is in the correct XML format and returns a Boolean value.

  Parameter: text

  Return type: Boolean.

  Example:

```
gaussdb=# SELECT xml_is_well_formed_document('<pg:foo xmlns:pg="http://postgresql.org/
stuff">bar</pg:foo>');
 xml_is_well_formed_document
-----------------------------
 t
(1 row)
```

- xml_is_well_formed_content(text)

  Description: Checks whether the text is in the correct XML format and returns a Boolean value.

  Parameter: text

  Return type: Boolean.

  Example:

```
gaussdb=# select xml_is_well_formed_content('k');
 xml_is_well_formed_content
----------------------------
 t
(1 row)
```


- xpath(xpath, xml [, nsarray])

  Description: Calculates an XPath 1.0 expression, for example, xpath (a text value), on XML data. It returns an array of XML values corresponding to the node set generated by the XPath expression. If the XPath expression returns a scalar value rather than a node set, a single-element array is returned.

  The second parameter must be a well formed XML document. Note that it must have a single root node element.

  The optional third parameter of the function is an array of namespace mappings. This array should be a two-dimensional text array with the length of the second axis being equal to 2 (that is, it should be an array of arrays, each of which consists of exactly 2 elements). The first element of each array entry is the namespace name (alias), and the second element is the namespace URI. It is not required that aliases provided in this array be the same as those being used in the XML document itself (in other words, both in the XML document and in the XPath function context, aliases are local).

  Return type: XML

  Example:

```
gaussdb=# SELECT xpath('/my:a/text()', '<my:a xmlns:my="http://example.com">test</
my:a>',ARRAY[ARRAY['my', 'http://example.com']]);
 xpath
--------
 {test}
(1 row)
```

- xpath_exists(xpath, xml [, nsarray])

  Description: This function is a special form of the xpath function. It returns a Boolean indicating whether the query is satisfied or not (specifically, whether it produces any value other than an empty node set), instead of returning the individual XML values that satisfy the XPath 1.0 expression. This function is equivalent to the standard XMLEXISTS predicate, but it also provides support for a namespace mapping parameter.

  Return type: Boolean.

  Example:

```
gaussdb=# SELECT xpath_exists('/my:a/text()', '<my:a xmlns:my="http://example.com">test</
my:a>',ARRAY[ARRAY['my', 'http://example.com']]);
 xpath_exists
--------------
 t
(1 row)
```


NOTE

The following XML function examples show the data you need to prepare:

```
gaussdb=# CREATE SCHEMA testxmlschema;
CREATE SCHEMA
gaussdb=# CREATE TABLE testxmlschema.test1 (a int, b text);
CREATE TABLE
gaussdb=# INSERT INTO testxmlschema.test1 VALUES (1, 'one'), (2, 'two'), (-1, null);
INSERT 0 3
gaussdb=# CREATE DATABASE test;
CREATE DATABASE

-- After the example is executed, run the following commands to delete the preceding data:
gaussdb=# DROP DATABASE test;
DROP DATABASE
gaussdb=# DROP TABLE testxmlschema.test1;
DROP TABLE
gaussdb=# DROP SCHEMA testxmlschema;
DROP SCHEMA
```

- query_to_xml(query text, nulls boolean, tableforest boolean, targetns text)

  Description: This function maps the contents of a query to an XML schema document.

  Return type: XML

  Example:

```
gaussdb=# SELECT query_to_xml('SELECT * FROM testxmlschema.test1', false, false, '');
                    query_to_xml
-------------------------------------------------------------
 <table xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+
                                                             +
 <row>                                                       +
   <a>1</a>                                                  +
   <b>one</b>                                                +
 </row>                                                      +
                                                             +
 <row>                                                       +
   <a>2</a>                                                  +
   <b>two</b>                                                +
 </row>                                                      +
                                                             +
 <row>                                                       +
   <a>-1</a>                                                 +
 </row>                                                      +
                                                             +
 </table>                                                    +
(1 row)
```

- query_to_xmlschema(query text, nulls boolean, tableforest boolean, targetns text)

  Description: This function maps the contents of a query to an XML document and an XML schema document, and joins the two documents together.

  Return type: XML

  Example:

```
gaussdb=# SELECT query_to_xmlschema('SELECT * FROM testxmlschema.test1', false, false, '');
                                query_to_xmlschema
-----------------------------------------------------------------------------------------------
 <xsd:schema                                                                  +
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">                                    +
                                                                        +
 <xsd:simpleType name="INTEGER">                                              +
   <xsd:restriction base="xsd:int">                                      +
     <xsd:maxInclusive value="2147483647"/>                                    +
     <xsd:minInclusive value="-2147483648"/>                                   +
   </xsd:restriction>                                                    +
```



```
</xsd:simpleType>                                                    +
                                                                    +
<xsd:simpleType name="UDT.regression.pg_catalog.text">                +
  <xsd:restriction base="xsd:string">                                +
  </xsd:restriction>                                                +
</xsd:simpleType>                                                    +
                                                                    +
<xsd:complexType name="RowType">                                      +
  <xsd:sequence>                                                    +
    <xsd:element name="a" type="INTEGER" minOccurs="0"></xsd:element>  +
    <xsd:element name="b" type="UDT.regression.pg_catalog.text" minOccurs="0"></xsd:element>+
  </xsd:sequence>                                                   +
</xsd:complexType>                                                   +
                                                                    +
<xsd:complexType name="TableType">                                    +
  <xsd:sequence>                                                    +
    <xsd:element name="row" type="RowType" minOccurs="0" maxOccurs="unbounded"/>  +
  </xsd:sequence>                                                   +
</xsd:complexType>                                                   +
                                                                    +
<xsd:element name="table" type="TableType"/>                          +
                                                                    +
</xsd:schema>
(1 row)
```

- query_to_xml_and_xmlschema(query text, nulls boolean, tableforest boolean, targetns text)

  Description: This function maps the contents of a query to an XML document and an XML schema document, and joins the two documents together.

  Return type: XML

  Example:

```
gaussdb=# SELECT query_to_xml_and_xmlschema('SELECT * FROM testxmlschema.test1', true, true, '');
                                query_to_xml_and_xmlschema
-------------------------------------------------------------------------------------------------
<xsd:schema                                                          +
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">                      +
                                                                    +
<xsd:simpleType name="INTEGER">                                       +
  <xsd:restriction base="xsd:int">                                   +
    <xsd:maxInclusive value="2147483647"/>                            +
    <xsd:minInclusive value="-2147483648"/>                           +
  </xsd:restriction>                                                +
</xsd:simpleType>                                                    +
                                                                    +
<xsd:simpleType name="UDT.regression.pg_catalog.text">                +
  <xsd:restriction base="xsd:string">                                +
  </xsd:restriction>                                                +
</xsd:simpleType>                                                    +
                                                                    +
<xsd:complexType name="RowType">                                      +
  <xsd:sequence>                                                    +
    <xsd:element name="a" type="INTEGER" nillable="true"></xsd:element>  +
    <xsd:element name="b" type="UDT.regression.pg_catalog.text" nillable="true"></xsd:element>+
  </xsd:sequence>                                                   +
</xsd:complexType>                                                   +
                                                                    +
<xsd:element name="row" type="RowType"/>                              +
                                                                    +
</xsd:schema>                                                        +
                                                                    +
<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">           +
                                                                    +
  <a>1</a>                                                          +
  <b>one</b>                                                        +
</row>                                                              +
                                                                    +
<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">           +
```

```
                                                                +
  <a>2</a>                                                       +
  <b>two</b>                                                      +
</row>                                                          +
                                                                +
<row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">                          +
                                                                +
  <a>-1</a>                                                      +
  <b xsi:nil="true"/>                                             +
</row>                                                          +
                                                                +
(1 row)
```

- cursor_to_xml(cursor refcursor, count int, nulls boolean,tableforest boolean, targetns text)

  Description: This function maps the contents of a cursor to an XML document.

  Return type: XML

  Example:

```
gaussdb=# CURSOR xc WITH HOLD FOR SELECT * FROM testxmlschema.test1 ORDER BY 1, 2;
DECLARE CURSOR
gaussdb=# SELECT cursor_to_xml('xc'::refcursor, 5, false, true, '');
                    cursor_to_xml
-------------------------------------------------------------
 <row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+
                                                +
  <a>-1</a>                                      +
 </row>                                         +
                                                +
 <row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+
                                                +
  <a>1</a>                                       +
  <b>one</b>                                      +
 </row>                                         +
                                                +
 <row xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+
                                                +
  <a>2</a>                                       +
  <b>two</b>                                      +
 </row>                                         +
                                                +

(1 row)
```

- cursor_to_xmlschema(cursor refcursor, nulls boolean, tableforest boolean, targetns text)

  Description: This function maps the contents of a cursor to an XML schema document.

  Return type: XML

  Example:

```
gaussdb=# SELECT cursor_to_xmlschema('xc'::refcursor, true, false, '');
                                cursor_to_xmlschema
-----------------------------------------------------------------------------------------------
 <xsd:schema                                                                    +
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">                                      +
                                                                                +
 <xsd:simpleType name="INTEGER">                                                      +
  <xsd:restriction base="xsd:int">                                                   +
    <xsd:maxInclusive value="2147483647"/>                                            +
    <xsd:minInclusive value="-2147483648"/>                                           +
  </xsd:restriction>                                                               +
 </xsd:simpleType>                                                                  +
                                                                                +
 <xsd:simpleType name="UDT.regression.pg_catalog.text">                                 +
  <xsd:restriction base="xsd:string">                                                +
```



```
  </xsd:restriction>                                                    +
</xsd:simpleType>                                                         +
                                                                        +
<xsd:complexType name="RowType">                                          +
  <xsd:sequence>                                                        +
    <xsd:element name="a" type="INTEGER" nillable="true"></xsd:element>     +
    <xsd:element name="b" type="UDT.regression.pg_catalog.text" nillable="true"></xsd:element>+
  </xsd:sequence>                                                       +
</xsd:complexType>                                                        +
                                                                        +
<xsd:complexType name="TableType">                                        +
  <xsd:sequence>                                                        +
    <xsd:element name="row" type="RowType" minOccurs="0" maxOccurs="unbounded"/>      +
  </xsd:sequence>                                                       +
</xsd:complexType>                                                        +
                                                                        +
<xsd:element name="table" type="TableType"/>                              +
                                                                        +
</xsd:schema>
(1 row)
```

- schema_to_xml(schema name, nulls boolean, tableforest boolean, targetns text)

  Description: This function maps the contents of the entire schema to an XML document.

  Return type: XML

  Example:

```
gaussdb=# SELECT schema_to_xml('testxmlschema', false, true, '');
                        schema_to_xml
-----------------------------------------------------------------------
 <testxmlschema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+
                                                                     +
 <test1>                                                               +
                                                                     +
   <a>1</a>                                                            +
   <b>one</b>                                                            +
 </test1>                                                              +
                                                                     +
 <test1>                                                               +
                                                                     +
   <a>2</a>                                                            +
   <b>two</b>                                                            +
 </test1>                                                              +
                                                                     +
 <test1>                                                               +
                                                                     +
   <a>-1</a>                                                           +
 </test1>                                                              +
                                                                     +
                                                                     +
 </testxmlschema>                                                          +
(1 row)
```

- schema_to_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text)

  Description: This function maps the contents of the entire schema to an XML schema document.

  Return type: XML

  Example:

```
gaussdb=# SELECT schema_to_xmlschema('testxmlschema', false, true, '');
                                    schema_to_xmlschema
-----------------------------------------------------------------------------------------------------
 <xsd:schema                                                                                    +
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">                                                 +
```



```
                                                                +
<xsd:simpleType name="INTEGER">                                   +
  <xsd:restriction base="xsd:int">                              +
    <xsd:maxInclusive value="2147483647"/>                        +
    <xsd:minInclusive value="-2147483648"/>                       +
  </xsd:restriction>                                          +
</xsd:simpleType>                                               +
                                                                +
<xsd:simpleType name="UDT.t1.pg_catalog.text">                    +
  <xsd:restriction base="xsd:string">                           +
  </xsd:restriction>                                          +
</xsd:simpleType>                                               +
                                                                +
<xsd:complexType name="SchemaType.t1.testxmlschema">                                +
  <xsd:sequence>                                                +
    <xsd:element name="test1" type="RowType.t1.testxmlschema.test1" minOccurs="0"
maxOccurs="unbounded"/>+
  </xsd:sequence>                                             +
</xsd:complexType>                                              +
                                                                +
<xsd:element name="testxmlschema" type="SchemaType.t1.testxmlschema"/>
+
                                                                +
</xsd:schema>
(1 row)
```

- schema_to_xml_and_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text)

  Description: This function maps the contents of the entire schema to an XML document and an XML schema document, and joins the two documents together.

  Return type: XML

  Example:

```
gaussdb=# SELECT schema_to_xml_and_xmlschema('testxmlschema', true, true, 'foo');
                                schema_to_xml_and_xmlschema
-------------------------------------------------------------------------------------------------------------
<testxmlschema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="foo"
xsi:schemaLocation="foo #">+
                                                                +
                                                          +
<xsd:schema                                                     +
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"                                  +
    targetNamespace="foo"                                       +
    elementFormDefault="qualified">                               +
                                                                +
<xsd:simpleType name="INTEGER">                                   +
  <xsd:restriction base="xsd:int">                              +
    <xsd:maxInclusive value="2147483647"/>                        +
    <xsd:minInclusive value="-2147483648"/>                       +
  </xsd:restriction>                                          +
</xsd:simpleType>                                               +
                                                                +
<xsd:simpleType name="UDT.t1.pg_catalog.text">                    +
  <xsd:restriction base="xsd:string">                           +
  </xsd:restriction>                                          +
</xsd:simpleType>                                               +
                                                                +
<xsd:complexType name="SchemaType.t1.testxmlschema">                                +
  <xsd:sequence>                                                +
    <xsd:element name="test1" type="RowType.t1.testxmlschema.test1" minOccurs="0"
maxOccurs="unbounded"/>   +
  </xsd:sequence>                                             +
</xsd:complexType>                                              +
                                                                +
<xsd:element name="testxmlschema"
type="SchemaType.t1.testxmlschema"/>                      +
```

```
                                                                +
</xsd:schema>                                                        +
                                                                +
<test1>                                                           +
                                                                +
  <a>1</a>                                                           +
  <b>one</b>                                                          +
</test1>                                                          +
                                                                +
<test1>                                                           +
                                                                +
  <a>2</a>                                                           +
  <b>two</b>                                                          +
</test1>                                                          +
                                                                +
<test1>                                                           +
                                                                +
  <a>-1</a>                                                          +
  <b xsi:nil="true"/>                                                 +
</test1>                                                          +
                                                                +
                                                                +
</testxmlschema>                                                        +

(1 row)
```

- database_to_xml(nulls boolean, tableforest boolean, targetns text)

  Description: This function maps the contents of the entire database to an XML document.

  Return type: XML

  Example:

```
gaussdb=# SELECT database_to_xml(true, true, 'test');
                          database_to_xml
-------------------------------------------------------------------------
 <test xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="test">+
                                                                +
<dbe_x005F_xml>                                                         +
                                                                +
</dbe_x005F_xml>                                                        +
                                                                +
<dbe_x005F_xmldom>                                                        +
                                                                +
</dbe_x005F_xmldom>                                                       +
                                                                +
<dbe_x005F_xmlparser>                                                     +
                                                                +
</dbe_x005F_xmlparser>                                                    +
                                                                +
<public>                                                             +
                                                                +
</public>                                                            +
                                                                +
</test>                                                           +

(1 row)
```

- database_to_xmlschema(nulls boolean, tableforest boolean, targetns text)

  Description: This function maps the contents of the entire database to an XML schema document.

  Return type: XML

  Example:

```
gaussdb=# SELECT database_to_xmlschema(true, true, 'test');
                            database_to_xmlschema
-------------------------------------------------------------------------------------
 <xsd:schema                                                                +
```

```
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"                                  +
    targetNamespace="test"                                                        +
    elementFormDefault="qualified">                                               +
                                                                                  +
<xsd:complexType name="CatalogType.test">                                         +
  <xsd:all>                                                                       +
    <xsd:element name="dbe_x005F_xml" type="SchemaType.test.dbe_x005F_xml"/>      +
    <xsd:element name="dbe_x005F_xmldom" type="SchemaType.test.dbe_x005F_xmldom"/> +
    <xsd:element name="dbe_x005F_xmlparser" type="SchemaType.test.dbe_x005F_xmlparser"/>+
    <xsd:element name="public" type="SchemaType.test.public"/>                    +
  </xsd:all>                                                                      +
</xsd:complexType>                                                                +
                                                                                  +
<xsd:element name="test" type="CatalogType.test"/>                                +
                                                                                  +
</xsd:schema>
(1 row)
```

- database_to_xml_and_xmlschema(nulls boolean, tableforest boolean, targetns text)

  Description: This function maps the contents of the entire schema to an XML document and an XML schema document, and joins the two documents together.

  Return type: XML

  Example:

```
gaussdb=# SELECT database_to_xml_and_xmlschema(true, true, 'test');
                                database_to_xml_and_xmlschema
-----------------------------------------------------------------------------------------------------
 <test xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="test"
xsi:schemaLocation="test #">+
                                                                                  +
<xsd:schema                                                                       +
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"                                  +
    targetNamespace="test"                                                        +
    elementFormDefault="qualified">                                               +
                                                                                  +
<xsd:complexType name="CatalogType.test">                                         +
  <xsd:all>                                                                       +
    <xsd:element name="dbe_x005F_xml" type="SchemaType.test.dbe_x005F_xml"/>      +
    <xsd:element name="dbe_x005F_xmldom"
type="SchemaType.test.dbe_x005F_xmldom"/>                                         +
    <xsd:element name="dbe_x005F_xmlparser"
type="SchemaType.test.dbe_x005F_xmlparser"/>                                      +
    <xsd:element name="public" type="SchemaType.test.public"/>                    +
  </xsd:all>                                                                      +
</xsd:complexType>                                                                +
                                                                                  +
<xsd:element name="test" type="CatalogType.test"/>                                +
                                                                                  +
</xsd:schema>                                                                     +
                                                                                  +
<dbe_x005F_xml>                                                                   +
                                                                                  +
</dbe_x005F_xml>                                                                  +
                                                                                  +
<dbe_x005F_xmldom>                                                                +
                                                                                  +
</dbe_x005F_xmldom>                                                               +
                                                                                  +
<dbe_x005F_xmlparser>                                                             +
                                                                                  +
</dbe_x005F_xmlparser>                                                            +
                                                                                  +
<public>                                                                          +
                                                                                  +
</public>                                                                         +
                                                                                  +
```



```
</test>                                                                 +

(1 row)
```

- table_to_xml(tbl regclass, nulls boolean, tableforest boolean, targetns text)

  Description: This function maps the contents of a relational table to an XML document.

  Return type: XML

  Example:

```
gaussdb=# SELECT table_to_xml('testxmlschema.test1', false, false, '');
                      table_to_xml
-------------------------------------------------------------
 <test1 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+
                                                            +
 <row>                                                      +
   <a>1</a>                                                 +
   <b>one</b>                                               +
 </row>                                                     +
                                                            +
 <row>                                                      +
   <a>2</a>                                                 +
   <b>two</b>                                               +
 </row>                                                     +
                                                            +
 <row>                                                      +
   <a>-1</a>                                                +
 </row>                                                     +
                                                            +
 </test1>                                                   +

(1 row)
```

- table_to_xmlschema(tbl regclass, nulls boolean, tableforest boolean, targetns text)

  Description: This function maps the contents of a relational table to an XML schema document.

  Return type: XML

  Example:

```
gaussdb=# SELECT table_to_xmlschema('testxmlschema.test1', false, false, '');
                                        table_to_xmlschema
-------------------------------------------------------------------------------------------------------------------
 <xsd:schema                                                                                          +
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">                                                             +
                                                                                                     +
 <xsd:simpleType name="INTEGER">                                                                          +
   <xsd:restriction base="xsd:int">                                                                          +
     <xsd:maxInclusive value="2147483647"/>                                                                   +
     <xsd:minInclusive value="-2147483648"/>                                                                  +
   </xsd:restriction>                                                                                    +
 </xsd:simpleType>                                                                                         +
                                                                                                     +
 <xsd:simpleType name="UDT.regression.pg_catalog.text">                                                                +
   <xsd:restriction base="xsd:string">                                                                       +
   </xsd:restriction>                                                                                    +
 </xsd:simpleType>                                                                                         +
                                                                                                     +
 <xsd:complexType name="RowType.regression.testxmlschema.test1">
+
   <xsd:sequence>                                                                                        +
     <xsd:element name="a" type="INTEGER" minOccurs="0"></
xsd:element>                                        +
     <xsd:element name="b" type="UDT.regression.pg_catalog.text" minOccurs="0"></
xsd:element>                  +
   </xsd:sequence>                                                                                       +
```



```
</xsd:complexType>                                                                  +
                                                                                    +
<xsd:complexType name="TableType.regression.testxmlschema.test1">
+
  <xsd:sequence>                                                                    +
    <xsd:element name="row" type="RowType.regression.testxmlschema.test1" minOccurs="0"
maxOccurs="unbounded"/>+
  </xsd:sequence>                                                                   +
</xsd:complexType>                                                                  +
                                                                                    +
<xsd:element name="test1"
type="TableType.regression.testxmlschema.test1"/>                                   +
                                                                                    +
</xsd:schema>
(1 row)
```

- table_to_xml_and_xmlschema(tbl regclass, nulls boolean, tableforest boolean, targetns text)

  Description: This function maps the contents of a relational table to an XML document and an XML schema document, and joins the two documents together.

  Return type: XML

  Example:

```
gaussdb=# SELECT table_to_xml_and_xmlschema('testxmlschema.test1', false, false, '');
                                  table_to_xml_and_xmlschema
-------------------------------------------------------------------------------------------------------------
<test1 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="#">                +
                                                                                    +
<xsd:schema                                                                           +
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">                                                 +
                                                                                    +
<xsd:simpleType name="INTEGER">                                                                 +
  <xsd:restriction base="xsd:int">                                                    +
    <xsd:maxInclusive value="2147483647"/>                                                      +
    <xsd:minInclusive value="-2147483648"/>                                                     +
  </xsd:restriction>                                                                +
</xsd:simpleType>                                                                     +
                                                                                    +
<xsd:simpleType name="UDT.regression.pg_catalog.text">                                                +
  <xsd:restriction base="xsd:string">                                                   +
  </xsd:restriction>                                                                +
</xsd:simpleType>                                                                     +
                                                                                    +
<xsd:complexType name="RowType.regression.testxmlschema.test1">
+
  <xsd:sequence>                                                                    +
    <xsd:element name="a" type="INTEGER" minOccurs="0"></
xsd:element>                                      +
    <xsd:element name="b" type="UDT.regression.pg_catalog.text" minOccurs="0"></
xsd:element>                  +
  </xsd:sequence>                                                                   +
</xsd:complexType>                                                                  +
                                                                                    +
<xsd:complexType name="TableType.regression.testxmlschema.test1">
+
  <xsd:sequence>                                                                    +
    <xsd:element name="row" type="RowType.regression.testxmlschema.test1" minOccurs="0"
maxOccurs="unbounded"/>+
  </xsd:sequence>                                                                   +
</xsd:complexType>                                                                  +
                                                                                    +
<xsd:element name="test1"
type="TableType.regression.testxmlschema.test1"/>                                   +
                                                                                    +
</xsd:schema>                                                                         +
                                                                                    +
```

```
<row>                                                                +
  <a>1</a>                                                            +
  <b>one</b>                                                            +
</row>                                                               +
                                                                   +
<row>                                                                +
  <a>2</a>                                                            +
  <b>two</b>                                                            +
</row>                                                               +
                                                                   +
<row>                                                                +
  <a>-1</a>                                                           +
</row>                                                               +
                                                                   +
</test1>                                                             +

(1 row)
```

NOTE

- For XPath-related functions, only XPath() and XPath_exists() are supported. These functions use the XPath language to query XML files and depend on the Libxml2 library, which is provided only in XPath1.0. Therefore, only XPath 1.0 is supported.
- The XQuery, XML extension, and XSLT functions are not supported.

- getclobval(xml)

  Description: Converts an XML type to a CLOB type. This function is valid only when parameters **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s4**.

  Parameter: The input parameter is of the XML type.

  Return type: CLOB

  Example:

```
gaussdb=# SET a_format_version='10c';
SET
gaussdb=# SET a_format_dev_version='s4';
SET
gaussdb=# DECLARE
 xmldata xml;
 result clob;
BEGIN
 xmldata := '<a>123</a>';
 result := getclobval(xmldata);
 RAISE NOTICE 'result is : %',result;
END;
/
NOTICE:  result is : <a>123</a>
gaussdb=# SELECT getclobval(xmlparse(document '<a>123</a>'));
 getclobval
------------
 <a>123</a>
(1 row)
```

- getstringval(xml)

  Description: Converts an XML type to a string. This function is valid only when parameters **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s4**.

  Parameter: The input parameter is of the XML type.

  Return type: VARCHAR2

  Example:

```
gaussdb=# SET a_format_version='10c';
SET
gaussdb=# SET a_format_dev_version='s4';
```

```
SET
gaussdb=# DECLARE
 xmldata xml;
 result varchar2;
BEGIN
 xmldata := '<a>123<b>456</b></a>';
 result := getstringval(xmldata);
 RAISE NOTICE 'result is : %',result;
END;
/
NOTICE:  result is : <a>123<b>456</b></a>
gaussdb=# SELECT getstringval(xmlparse(document '<a>123<b>456</b></a>'));
    getstringval
----------------------
 <a>123<b>456</b></a>
(1 row)
```

- xmlsequence(xml)

  Description: Converts an XML parameter into an array of the XMLTYPE type. Each array element is an XMLTYPE object. The input parameter of this function cannot be null and must be a valid XML document. If the input parameter does not meet the requirements, the function returns a null value or throws an exception. This function can be used to process multiple child nodes in an XML document or split an XML document into multiple fragments.

  Parameter: XML type.

  Return type: Array of the XMLType type.

  Example 1: If you want to convert this document into an array containing three elements, each of which is a book node, use the following statement:

```
gaussdb=# SELECT xmlsequence(xml('<books><book><title>The Catcher in the Rye</
title><author>J.D. Salinger</author><year>1951</year></book><book><title>1984</
title><author>George Orwell</author><year>1949</year></book><book><title>The Hitchhiker''s
Guide to the Galaxy</title><author>Douglas Adams</author><year>1979</year></book></books>'));
                    xmlsequence
------------------------------------------------------------
 {"<books>                                          +
    <book>                                         +
      <title>The Catcher in the Rye</title>           +
      <author>J.D. Salinger</author>                 +
      <year>1951</year>                              +
    </book>                                        +
    <book>                                         +
      <title>1984</title>                            +
      <author>George Orwell</author>                    +
      <year>1949</year>                              +
    </book>                                        +
    <book>                                         +
      <title>The Hitchhiker's Guide to the Galaxy</title>+
      <author>Douglas Adams</author>                    +
      <year>1979</year>                              +
    </book>                                        +
  </books>"}
(1 row)
```

  NOTE

  If the XML file inputting the xmlsequence function contains double quotation marks, the result of the xmlsequence function contains escape characters of double quotation marks when you query the result of the xmlsequence function independently. The use of the xmlsequence function is not affected.

7.6.43 Functions of the XMLType Type

- createxml(varchar2 [,varchar2 ,numeric ,numeric])

  Description: Statically creates the XMLType type. The input parameters are of the varchar2 type.

  Parameters: The first parameter is the character string to be converted to XMLType (mandatory column). The second parameter is the optional schema URL used to make the input comply with the specified schema (optional column, which is empty by default and does not take effect currently). The third parameter is the flag indicating whether the instance is valid according to the given XML schema (optional column, which is **0** by default and does not take effect currently). The fourth parameter is the flag indicating whether the instance is well-formed (optional column, which is **0** by default and does not take effect currently).

  Return type: XMLType

  Example:

```
gaussdb=# SELECT createxml('<a>123</a>');
 createxml
---------------
 <a>123</a>
(1 row)
```


NOTE

- Different from that in database A, in PL/SQL, createxml allows input parameters to be empty strings and returns **NULL**.
- For character encoding, only UTF-8, GBK, ZHS16GBK, and LATIN1 to LATIN10 are supported, and the **version** field can only be set to **1.x**.
- The createxml function can be invoked using the xmltype.createxml() syntax.

  Example:

```
gaussdb=# SELECT xmltype.createxml('<a>123</a>');
 createxml
------------
 <a>123</a>
(1 row)
```

- In this chapter, the function whose input parameter is xmltype() can be invoked in xmltype().func() mode. The XMLType type returned by a function is transferred to the next function as the input parameter. This syntax supports multi-layer nesting. (If the input parameter is defined as XMLType by a user, this syntax is not supported.)

  Example:

```
gaussdb=# select xmltype('<a>123<b>456</b></a>').extract('/a/b').getstringval();
 xmltypefunc
-------------
 <b>456</b>
(1 row)
```

  The actual effect of the preceding example is the same as that of the following function nesting:

```
gaussdb=# select getstringval(extractxml(xmltype('<a>123<b>456</b></a>'),'/a/b'));
 getstringval
--------------
 <b>456</b>
(1 row)
```

- In a stored procedure, variables of the XMLType type can invoke functions in a.func() mode. This syntax supports one-layer nesting.

  Example:

```
gaussdb=# declare
        a xmltype;
        b varchar2;
begin
        a:=xmltype('<a>123<b>456</b></a>');
        b:=a.getstringval();
        RAISE NOTICE 'xmltype_str is : %',b;
end;
/
NOTICE:  xmltype_str is : <a>123<b>456</b></a>
```

- createxml(clob [,varchar2 ,numeric ,numeric])

  Description: Statically creates the XMLType type. The input parameters are of the CLOB type.

  Parameters: The first parameter is the CLOB to be converted to XMLType (mandatory column). The second parameter is the optional schema URL used to make the input comply with the specified schema (optional column, which is empty by default and does not take effect currently). The third parameter is the flag indicating whether the instance is valid according to the given XML schema (optional column, which is **0** by default and does not take effect currently). The fourth parameter is the flag indicating whether the instance is well-formed (optional column, which is **0** by default and does not take effect currently).

  Return type: XMLType

Example:

```
gaussdb=# declare
 xmltype_clob clob;
 xmltype_obj xmltype;
 xmltype_str varchar2(1000);
begin
 xmltype_clob := '<a>123</a>';
 xmltype_obj := createxml(xmltype_clob);
 xmltype_str := xmltype_obj.getstringval();
 RAISE NOTICE 'xmltype_str is : %',xmltype_str;
end;
/
NOTICE:  xmltype_str is : <a>123</a>
```

NOTE

The maximum size of the input parameter of the CLOB type is 1 GB minus 1 byte.

- createxml(blob, numeric [,varchar2 ,numeric ,numeric])

  Description: Statically creates the XMLType type. The input parameters are of the BLOB type.

  Parameters: The first parameter is the BLOB to be converted to XMLType (mandatory column). The second parameter is the character set ID of the input XML data (mandatory column). The third parameter is the optional schema URL used to make the input comply with the specified schema (optional column, which is empty by default and does not take effect currently). The fourth parameter is the flag indicating whether the instance is valid according to the given XML schema (optional column, which is **0** by default and does not take effect currently). The fifth parameter is the flag indicating whether the instance is well-formed (optional column, which is **0** by default and does not take effect currently).

  Return type: XMLType

  Example:

```
gaussdb=# declare
 xmltype_blob blob;
 xmltype_obj xmltype;
 xmltype_str varchar2(1000);
begin
 xmltype_blob := xmltype('<a>123</a>').getblobval(7);
 xmltype_obj := createxml(xmltype_blob,7);
 xmltype_str := xmltype_obj.getstringval();
 RAISE NOTICE 'xmltype_str is : %',xmltype_str;
end;
/
NOTICE:  xmltype_str is : <?xml version="1.0" encoding="UTF8"?>
<a>123</a>
```

NOTE

- The maximum size of input parameters of the BLOB type is 256 MB minus 1 byte.
- The character set ID ranges from 1 to 43.

- getblobval(xmltype, numeric)

  Description: Converts the XMLType type to the BLOB type. The xmltype().func() method can be invoked.

  Parameters: The first parameter is of the XMLType type, and the second parameter is the ID of the target character set to be converted.

  Return type: BLOB

  Example:


```
gaussdb=# SELECT getblobval(xmltype('<asd/>'),7);
                                   getblobval
--------------------------------------------------------------------------------------------

3C3F786D6C2076657273696F6E3D22312E302220656E636F64696E673D2255544638223F3E0A3C6173
642F3E
(1 row)
```

xmltype ().func ():

```
gaussdb=# select xmltype('<asd/>').getblobVal(7);
                                   xmltypefunc
--------------------------------------------------------------------------------------------

3C3F786D6C2076657273696F6E3D22312E302220656E636F64696E673D2255544638223F3E0A3C6173
642F3E
(1 row)
```

NOTE

The maximum length of the input parameter of the XMLType type is 256 MB minus 1 byte.

- getclobval(xmltype)

  Description: Converts the XMLType type to the CLOB type. The xmltype().func() method can be invoked.

  Parameter: The input parameter is of the XMLType type.

  Return type: CLOB

  Example:

```
gaussdb=# SELECT getclobval(xmltype('<a>123</a>'));
 getclobval
--------------
 <a>123</a>
(1 row)
```

  xmltype ().func ():

```
gaussdb=# SELECT xmltype('<a>123</a>').getclobval();
 xmltypefunc
--------------
 <a>123</a>
(1 row)
```

- getnumberval(xmltype)

  Description: Converts the XMLType type to the numeric type. The xmltype().func() method can be invoked.

  Parameter: The input parameter is of the XMLType type.

  Return type: numeric

  Example:

```
gaussdb=# SELECT getnumberval(xmltype('<a>123</a>').extract('/a/text()'));
 getnumberval
--------------
          123
(1 row)
```

  xmltype ().func ():

```
gaussdb=# SELECT xmltype('<a>123</a>').extract('/a/text()').getnumberval();
 xmltypefunc
--------------
          123
(1 row)
```

- isfragment(xmltype)

  Description: Returns a result indicating whether the XMLType type is fragment (1) or document (0). The xmltype().func() method can be invoked.

Parameter: The input parameter is of the XMLType type.

Return type: numeric

Example:

```
gaussdb=# SELECT isfragment(xmltype('<a>123</a>'));
 isfragment
--------------
          0
(1 row)
```

xmltype ().func ():

```
gaussdb=# SELECT xmltype('<a>123</a>').isfragment();
 xmltypefunc
--------------
          0
(1 row)
```

- xmltype(varchar2 [,varchar2,numeric ,numeric])

  Description: Creates the XMLType type from the varchar2 type.

  Parameters: The first parameter is the character string to be converted to XMLType (mandatory column). The second parameter is the optional schema URL used to make the input comply with the specified schema (optional column, which is empty by default and does not take effect currently). The third parameter is the flag indicating whether the instance is valid according to the given XML schema (optional column, which is **0** by default and does not take effect currently). The fourth parameter is the flag indicating whether the instance is well-formed (optional column, which is **0** by default and does not take effect currently).

  Return type: XMLType

  Example:

  ```
  gaussdb=# SELECT xmltype('<a>123</a>');
   xmltype
  --------------
   <a>123</a>
  (1 row)
  ```

  NOTE

  - Different from that in database A, in PL/SQL, XMLType allows input parameters to be empty strings and returns **NULL**.
  - For character encoding, only UTF-8, GBK, ZHS16GBK, and LATIN1 to LATIN10 are supported, and the **version** field can only be set to **1.x**.

- xmltype(clob [,varchar2 ,numeric ,numeric])

  Description: Creates the XMLType type from the CLOB type.

  Parameters: The first parameter is the CLOB to be converted to XMLType (mandatory column). The second parameter is the optional schema URL used to make the input comply with the specified schema (optional column, which is empty by default and does not take effect currently). The third parameter is the flag indicating whether the instance is valid according to the given XML schema (optional column, which is **0** by default and does not take effect currently). The fourth parameter is the flag indicating whether the instance is well-formed (optional column, which is **0** by default and does not take effect currently).

  Return type: XMLType

  Example:

  ```
  gaussdb=# declare
   xmltype_clob clob;
  ```



```
 xmltype_obj xmltype;
 xmltype_str varchar2(1000);
begin
 xmltype_clob := '<a>123</a>';
 xmltype_obj := xmltype(xmltype_clob);
 xmltype_str := xmltype_obj.getstringval();
 RAISE NOTICE 'xmltype_str is : %',xmltype_str;
end;
/
NOTICE:  xmltype_str is : <a>123</a>
```

NOTE

The maximum size of the input parameter of the CLOB type is 1 GB minus 1 byte.

- xmltype(blob, numeric [,varchar2 ,numeric ,numeric])

  Description: Creates the XMLType type from the BLOB type.

  Parameters: The first parameter is the BLOB to be converted to XMLType (mandatory column). The second parameter is the character set ID of the input XML data. The third parameter is the optional schema URL used to make the input comply with the specified schema (optional column, which is empty by default and does not take effect currently). The fourth parameter is the flag indicating whether the instance is valid according to the given XML schema (optional column, which is **0** by default and does not take effect currently). The fifth parameter is the flag indicating whether the instance is well-formed (optional column, which is **0** by default and does not take effect currently).

  Return type: XMLType

  Example:

```
gaussdb=# declare
 xmltype_blob blob;
 xmltype_obj xmltype;
 xmltype_str varchar2(1000);
begin
 xmltype_blob := getblobval(createxml('<a>123</a>'),7);
 xmltype_obj := xmltype(xmltype_blob,7);
 xmltype_str := xmltype_obj.getstringval();
 RAISE NOTICE 'xmltype_str is : %',xmltype_str;
end;
/
NOTICE:  xmltype_str is : <?xml version="1.0" encoding="UTF8"?>
<a>123</a>
```

NOTE

- The maximum size of input parameters of the BLOB type is 256 MB minus 1 byte.
- The character set ID ranges from 1 to 42.

- getstringval(xmltype)

  Description: Converts xmltype into a string.

  Parameter: XMLType to be converted.

  Return type: varchar2

  The getstringval function can be invoked in either of the following ways:

  Example 1:

```
gaussdb=# SELECT getstringval('<a>123<b>456</b></a>');
    getstringval
----------------------
 <a>123<b>456</b></a>
(1 row)
```


Example 2: The invoking mode is compatible with the ORA syntax.

```
gaussdb=# SELECT xmltype('<a>123<b>456</b></a>').getstringval();
    xmltypefunc
----------------------
 <a>123<b>456</b></a>
(1 row)
```

- getrootelement(xmltype)

  Description: Gets the root element of the XMLType type.

  Parameter: XMLType whose root element is to be obtained.

  Return type: varchar2

  The getrootelement function can be invoked in either of the following ways:

  Example 1:

```
gaussdb=# SELECT getrootelement('<a>123<b>456</b></a>');
 getrootelement
----------------
 a
(1 row)
```

  Example 2: The invoking mode is compatible with the ORA syntax.

```
gaussdb=# SELECT xmltype('<a>123<b>456</b></a>').getrootelement();
 xmltypefunc
-------------
 a
(1 row)
```

- getnamespace(xmltype)

  Description: Gets the namespace of the XMLType top-level element.

  Parameter: XMLType whose namespace is to be obtained.

  Return type: varchar2

  The getnamespace function can be invoked in either of the following ways:

  Example 1:

```
gaussdb=# SELECT getnamespace('<c:a xmlns:c="asd">123<d:b xmlns:d="qwe">456</d:b></c:a>');
 getnamespace
--------------
 asd
(1 row)
```

  Example 2: The invoking mode is compatible with the ORA syntax.

```
gaussdb=# SELECT xmltype('<c:a xmlns:c="asd">123<d:b xmlns:d="qwe">456</d:b></
c:a>').getnamespace();
 xmltypefunc
-------------
 asd
(1 row)
```

- existsnode(xmltype, varchar2[, varchar2])

  Description: Determines whether the XML node exists in XMLType based on the XPath expression. If the XML node exists, **1** is returned. Otherwise, **0** is returned.

  Parameters: XMLType to be queried, path of the XPath node to be queried, and namespace of the XPath path (If the input parameter has a namespace, aliases must be defined for both the XPath and namespace, as shown in example 3.)

  Return type: numeric

  The existsnode function can be invoked in either of the following ways:

Example 1:

```
gaussdb=# SELECT existsnode('<a>123<b>456</b></a>','/a/b');
 existsnode
------------
          1
(1 row)
```

Example 2: The invoking mode is compatible with the ORA syntax.

```
gaussdb=# SELECT xmltype('<a>123<b>456</b></a>').existsnode('/a/b');
 xmltypefunc
-------------
           1
(1 row)
```

Example 3

```
gaussdb=# SELECT existsnode('<a:b xmlns:a="asd">123<c>456</c></a:b>','/a:b/c','xmlns:a="asd"');
 existsnode
------------
          1
(1 row)
```

Example 4: The invoking mode is compatible with the ORA syntax.

```
gaussdb=# SELECT xmltype('<a:b xmlns:a="asd">123<c>456</c></a:b>').existsnode('/a:b/
c','xmlns:a="asd"');
 xmltypefunc
-------------
           1
(1 row)
```

- extractxml(xmltype, varchar2[, varchar2])

  Description: Checks whether an XML node exists in the given XMLType based on the XPath expression. If yes, the XMLType containing the node is returned. If no, **NULL** is returned. The return value can be inserted into a table of the XMLType type.

  Parameters: XMLType to be queried, path of the XPath node to be queried, and namespace of the XPath path (If the input parameter has a namespace, aliases must be defined for both the XPath and namespace, as shown in example 3.)

  Return type: XMLType

  The extractxml function can be invoked in either of the following ways:

  Example 1:

```
gaussdb=# SELECT extractxml('<a>123<b>456</b></a>','/a/b');
 extractxml
------------
 <b>456</b>
(1 row)
```

  Example 2: The invoking mode is compatible with the ORA syntax.

```
gaussdb=# SELECT xmltype('<a>123<b>456</b></a>').extract('/a/b');
 xmltypefunc
-------------
 <b>456</b>
(1 row)

gaussdb=# SELECT xmltype('<a>123<b>456</b></a>').extractxml('/a/b');
 xmltypefunc
-------------
 <b>456</b>
(1 row)
```

  Example 3

```
gaussdb=# SELECT extractxml('<a:b xmlns:a="asd">123<c>456</c></a:b>','/a:b','xmlns:a="asd"');
              extractxml
```



```
----------------------------------------
 <a:b xmlns:a="asd">123<c>456</c></a:b>
(1 row)
```

Example 4: The invoking mode is compatible with the ORA syntax.

```
gaussdb=# SELECT xmltype('<a:b xmlns:a="asd">123<c>456</c></a:b>').extract('/a:b','xmlns:a="asd"');
            xmltypefunc
----------------------------------------
 <a:b xmlns:a="asd">123<c>456</c></a:b>
(1 row)

gaussdb=# SELECT xmltype('<a:b xmlns:a="asd">123<c>456</c></a:b>').extractxml('/
a:b','xmlns:a="asd"');
            xmltypefunc
----------------------------------------
 <a:b xmlns:a="asd">123<c>456</c></a:b>
(1 row)
```

- extractvalue(xmltype | xml, varchar2[, varchar2])

  Description: Extracts the value of an XPath expression from an XML file based on the XPath expression (only XPath 1.0 is supported). The result of the XPath expression must be a single node and must be a text node, attribute, or element. If the XPath expression contains an element expression, the element must have a text node as a child node. The function returns the text. If the result is an attribute, the function returns the value of the attribute.

  Parameters: **xmltype | xml**: XML text to be queried; **varchar2**: XPath expression to be queried (XPath node path); **[, varchar2]** (optional): Namespace of the XPath node path. (If the input parameter contains a namespace with an alias, you need to define an alias for both the XPath expression and namespace. If the default namespace is used, you do not need to define an alias, as shown in example 3.)

  Return type: varchar2

  The extractvalue function can be invoked in two ways: input without namespace and input with namespace.

  Example 1: The input does not contain a namespace.

```
gaussdb=# SELECT EXTRACTVALUE(xmltype('<book><title>Harry Potter</title><author>J.K. Rowling</
author></book>'), '/book/title') AS book_title;
  book_title
--------------
 Harry Potter
(1 row)
```

  Example 2: The input contains a namespace.

```
gaussdb=# SELECT EXTRACTVALUE(xmltype('<ns:book xmlns:ns="http://
www.example.com"><ns:title>Harry Potter</ns:title><ns:author>J.K. Rowling</ns:author></ns:book>'),
'/ns:book/ns:title', 'xmlns:ns="http://www.example.com"') AS book_title;
  book_title
--------------
 Harry Potter
(1 row)
```

  Example 3: The input contains multiple namespaces.

```
gaussdb=# SELECT EXTRACTVALUE(xmltype('<ns:book xmlns:ns="http://www.example.com"
xmlns:ff="http://www.ff.com"><ff:title>Harry Potter</ff:title><ns:author>J.K. Rowling</ns:author></
ns:book>'), '/ns:book/ff:title', 'xmlns:ns="http://www.example.com" xmlns:ff="http://www.ff.com"') AS
book_title;
  book_title
--------------
 Harry Potter
(1 row)
gaussdb=# SELECT EXTRACTVALUE(xmltype('<store><book xmlns="abc"><root xmlns="abcd">mike</
root><root>mikeab</root></book><root xmlns="abcd">mikedwsa</root></store>'),
```



```
'//root', 'xmlns="abc" xmlns:ns2="abc1" xmlns="abcd"') FROM dual;
 extractvalue
--------------
 mikeab
(1 row)
```

NOTE

- If the input contains multiple namespaces, they can be separated by one or more spaces (or newline characters). However, the namespace expression is in the **xmlns:***Name*="*Namespace*" format and the default namespace rule is in the **xmlns='***URL*' format.
- This function is compatible with the XMLType expression function, but the node value returned by the XMLType text must be unique.
- The XPath expression supports only XPath 1.0.
- Currently, the namespace URL in the XML text cannot be a space, and the namespace URL in the namespace expression cannot be a space.
- In the default namespace scenario, the default namespace declared first in the namespace expression is the default namespace of the current XML text.

- xmlsequence(xmltype)

  Description: Converts an XMLTYPE parameter into an array of the XMLTYPE type. Each array element is an XMLTYPE object. The input parameter of this function cannot be null and must be a valid XML document. If the input parameter does not meet the requirements, the function returns a null value or throws an exception. This function can be used to process multiple child nodes in an XML document or split an XML document into multiple fragments.

  Parameter: The input parameter is of the XMLType type.

  Return value type: array of the XMLType type

  Example 1: If you want to convert this document into an array containing three elements, each of which is a book node, use the following statement:

```
gaussdb=# SELECT xmlsequence(xmltype('<books><book><title>The Catcher in the Rye</
title><author>J.D. Salinger</author><year>1951</year></book><book><title>1984</
title><author>George Orwell</author><year>1949</year></book><book><title>The Hitchhiker''s
Guide to the Galaxy</title><author>Douglas Adams</author><year>1979</year></book></books>'));
                        xmlsequence
------------------------------------------------------------
 {"<books>                                      +
    <book>                                      +
      <title>The Catcher in the Rye</title>           +
      <author>J.D. Salinger</author>                  +
      <year>1951</year>                             +
    </book>                                     +
    <book>                                      +
      <title>1984</title>                         +
      <author>George Orwell</author>                    +
      <year>1949</year>                             +
    </book>                                     +
    <book>                                      +
      <title>The Hitchhiker's Guide to the Galaxy</title>+
      <author>Douglas Adams</author>                    +
      <year>1979</year>                             +
    </book>                                     +
  </books>"}
(1 row)
```

  Example 2: If you want to extract the title and author of each book from this array, use the following statement:

```
gaussdb=# SELECT unnest(xmlsequence(xmltype('<books><book><title>The Catcher in the Rye</
title><author>J.D. Salinger</author><year>1951</year></book><book><title>1984</
title><author>George Orwell</author><year>1949</year></book><book><title>The Hitchhiker''s
```



```
Guide to the Galaxy</title><author>Douglas Adams</author><year>1979</year></book></
books>').extract('//title/text()')) AS title
    , unnest(xmlsequence(xmltype('<books><book><title>The Catcher in the Rye</title><author>J.D.
Salinger</author><year>1951</year></book><book><title>1984</title><author>George Orwell</
author><year>1949</year></book><book><title>The Hitchhiker''s Guide to the Galaxy</
title><author>Douglas Adams</author><year>1979</year></book></books>').extract('//author/
text()'))) AS author;
                        title                         |              author
------------------------------------------------------------------+-----------------------------------------
 The Catcher in the Rye1984The Hitchhiker's Guide to the Galaxy | J.D. SalingerGeorge OrwellDouglas
Adams
(1 row)
```

Example 3: If you want to convert the array into a JSON string, use the following statement:

```
gaussdb=# SELECT array_to_json(array_agg(row_to_json(t)))
FROM (
    SELECT unnest(xmlsequence(xmltype('<books><book><title>The Catcher in the Rye</
title><author>J.D. Salinger</author><year>1951</year></book><book><title>1984</
title><author>George Orwell</author><year>1949</year></book><book><title>The Hitchhiker''s
Guide to the Galaxy</title><author>Douglas Adams</author><year>1979</year></book></
books>').extract('//title/text()')) AS title
        , unnest(xmlsequence(xmltype('<books><book><title>The Catcher in the Rye</title><author>J.D.
Salinnger</author><year>1951</year></book><book><title>1984</title><author>George Orwell</
author><year>1949</year></book><book><title>The Hitchhiker''s Guide to the Galaxy</
title><author>Douglas Adams</author><year>1979</year></book></books>').extract('//author/
text()'))) AS author
) t;


array_to_json
-----------------------------------------------------------------------------------------------------------
 [{"title":"The Catcher in the Rye1984The Hitchhiker's Guide to the Galaxy","author":"J.D.
SalinngerGeorge OrwellDouglas Adams"}]
(1 row)
```

NOTE

If the XML file inputting the xmlsequence function contains double quotation marks, the result of the xmlsequence function contains escape characters of double quotation marks when you query the result of the xmlsequence function independently. The use of the xmlsequence function is not affected.

7.6.44 Global PL/SQL Cache Functions

- invalidate_plsql_object(), invalidate_plsql_object(schema, objname, objtype);

  Description: Invalidates objects in the global PL/SQL cache. This function is available only when **enable_global_plsqlcache** is set to **on**. The user who calls this function must have the SYSADMIN permission.

  Parameter: This function is an overloaded function. If there is no input parameter, all global cache objects in all databases are invalidated.

  If **schema**, **objname**, and **objtype** are specified, the specified global cache object in the current database is invalidated. **schema** indicates the name of the schema to which the object belongs, **objname** indicates the object name, and **objtype** indicates the object type. If the object is a package, the value of **objtype** is **package**. If the object is a function or stored procedure, the value of **objtype** is **function**.

  Example:

  This function does not return the invalidation result. You can query the invalidation result in the gs_glc_memory_detail view. If the object is not invalidated, you can find it in valid state in the view. If the object is invalidated, there is no such state.

invalidate_plsql_object belongs to the pg_catalog schema. It can be invoked even if no schema is specified.

```
-- If the cache information of function f3 is displayed in the view, the function is valid.
gaussdb=# SELECT * FROM gs_glc_memory_detail WHERE type='func' or type='pkg';
 contextname | database | schema  | type | status  |        location         |   env    | usedsize
-------------+----------+---------+------+---------+-------------------------+----------+----------
 pkg1        | testdb | public  | pkg  | valid   | in_global_hash_table   |        0 |   184176
 f3          | testdb | public  | func | valid   | in_global_hash_table   |        0 |    47584
-- To invalidate function f3, specify the schema, function name, and type. Then, query the view again, the status of f3 is not valid.
gaussdb=# SELECT invalidate_plsql_object('public','f3','function');
 invalidate_plsql_object
-------------------------
(1 row)
-- To invalidate a package, set the parameters as follows:
gaussdb=# call pg_catalog.invalidate_plsql_object('public','pkg1','package');
 invalidate_plsql_object
-------------------------
(1 row)

-- If there is no input parameter, all cache objects will be invalidated.
gaussdb=# SELECT invalidate_plsql_object();
 invalidate_plsql_object
-------------------------
(1 row)
```

7.6.45 Pivot Table Functions

- tablefunc()

  Description: Extended API for processing table data, including pivot table functions. Only the system administrator can install extensions.

  NOTE

  By default, the extension is installed in the public schema. You are advised to install the extension in the user schema by running **create extension tablefunc [schema {user_schema}]**. The extended function is for internal use only. You are advised not to use it.

- crosstab(source_sql text [, N int])

  Description: Uses the result of **source_sql** as the source data to generate a pivot table.

  Return type: setof record

  Example:

```
gaussdb=# CREATE extension tablefunc;
CREATE EXTENSION
gaussdb=# CREATE TABLE cross_test(group_id text, id int, var text);
CREATE TABLE
gaussdb=# SELECT * FROM cross_test;
 group_id | id | var
----------+----+-----
(0 rows)

gaussdb=# SELECT * FROM crosstab('SELECT group_id, var FROM cross_test order by 1, 2;') AS c(group_ text, cat1 text, cat2 text, cat3 text);
 group_ | cat1 | cat2 | cat3
--------+------+------+------
(0 rows)
```

  NOTE

  N is a deprecated parameter and does not affect the function result.

- crosstabN(source_sql text)

  Description: Uses the result of **source_sql** as the source data to generate a pivot table with "*N*+1" columns. crosstabN is a group of functions, including crosstab2, crosstab3, and crosstab4.

  Return type: setof tablefunc_crosstab_N. tablefunc_crosstab_N includes tablefunc_crosstab_2, tablefunc_crosstab_3, and tablefunc_crosstab_4.

  Example:

```
-- If N in crossbatN(source_sql text) is set to 2, a pivot table with three columns is generated.
gaussdb=# CREATE extension tablefunc;
CREATE EXTENSION
gaussdb=# CREATE TABLE cross_test(group_id text, id int, var text);
CREATE TABLE
gaussdb=# SELECT * FROM crosstab2('SELECT group_id, var from cross_test ORDER BY 1, 2;');
 row_name | category_1 | category_2
----------+------------+------------
(0 rows)
```

- crosstab(source_sql text, category_sql text)

  Description: Uses the result of **source_sql** as the source data to generate a pivot table based on the result category of **category_sql**.

  Return type: setof record

  Example:

```
gaussdb=# CREATE extension tablefunc;
CREATE EXTENSION
gaussdb=# CREATE TABLE cross_test(group_id text, id int, var text);
CREATE TABLE
gaussdb=# SELECT * FROM crosstab('SLECT group_id, var FROM cross_test order by 1, 2;', 'SELECT
generate_series(1, 4)') AS c(group_ text, cat1 text, cat2 text, cat3 text, cat4 text);
 group_ | cat1 | cat2 | cat3 | cat4
--------+------+------+------+------
(0 rows)
```

7.6.46 UUID Functions

- sys_guid()

  Description: Generates and returns a 16-byte universally unique identifier (UUID). The generated UUID contains the IP address of the current node, generation timestamp, and random number.

  Parameter: none

  Return type: raw

  Example:

```
gaussdb=# SET a_format_version='10c';
SET
gaussdb=# SET a_format_dev_version='s5';
SET
gaussdb=# SELECT sys_guid();
              sys_guid
----------------------------------
 9010675E560CB33C1BDAFA163E378F87
 (1 row)
```

- uuid()

  Description: Returns a UUID defined in RFC 4122, ISO/IEF 9834-8:2005, and related standards. The identifier is a string of lowercase hexadecimal digits, which are divided into several groups: a group of 8 digits, three groups of 4 digits, and a group of 12 digits. A total of 32 digits represent 128 bits.

  Parameter: none

Return type: varchar

Example:

```
gaussdb=# SELECT uuid();
                 uuid
--------------------------------------
 dd8cbe92-1a25-013c-a514-e435c87e9182
(1 row)
```

- uuid_short()

  Description: Returns a short UUID under certain conditions. This identifier is a 64-bit unsigned integer.

  The returned value is unique when the following conditions are met:

  – The number of service nodes in the current cluster cannot exceed 256.

  – You cannot set the system time of the server host between node restarts.

  – The average number of uuid_short() calls per second between node restarts is less than 16 million.

  Parameter: none

  Return type: uint64

  Example:

  ```
  gaussdb=# SELECT uuid_short();
      uuid_short
  --------------------
   100440026956955649
  (1 row)
  ```

  NOTE

  For upgrade from a version to GaussDB Kernel 505.0.0, the uuid_short() function cannot be used if the upgrade is not committed.

7.6.47 SQL Statement Concurrency Control Function

- gs_add_workload_rule(rule_type, rule_name, databases, start_time, end_time, max_workload, option_val)

  Description: Creates an SQL statement concurrency control rule. Users must have the sysadmin permission.

  Parameters: For details, see **Table 7-164**.

  Return type: int8

Table 7-164 gs_add_workload_rule parameters

Parameter	Type	Description	Value Range
rule_type	text	Type of the concurrency control rule, which is case insensitive.	**"sqlid"**: Concurrency control is based on the unique SQL ID. **"select"**, **"insert"**, **"update"**, **"delete"**, and **"merge"**: Concurrency control is based on the query type and keyword. **"resource"**: Instance-level concurrency control is based on the system resource usage.
rule_name	name	Name of a concurrency control rule, which is used to search for the concurrency control rule.	Any character string or **NULL**.
databases	name[]	Array of database names for which the concurrency control rule takes effect. The value is case-sensitive.	List of names of created databases. The value can be **NULL**, indicating that the configuration takes effect in all databases. Currently, the database list takes effect only when **rule_type** is set to a query type because a unique SQL ID is bound to a database and belongs to only one database. The concurrency control rules based on resource usage take effect for instances, that is, all databases.
start_time	timestamptz	Start time of a concurrency control rule.	The value can be **NULL**, indicating that it takes effect from now on.
end_time	timestamptz	End time of a concurrency control rule.	The value can be **NULL**, indicating that the rule is always effective.

Parameter	Type	Description	Value Range
max_workload	int8	Maximum number of concurrent requests set in a concurrency control rule.	-
option_val	text[]	Supplementary information about the concurrency control rule.	It matches **rule_type**. The matching relationship is as follows: • **"sqlid"**: specifies the unique ID of the SQL statement whose concurrency is to be controlled and slow SQL control rule. The format is '{id=1234, time_limit=100, max_execute_time=500, max_iops=1}', in which **id** indicates the unique SQL ID and is mandatory. You can obtain it from the dbe_perf.statement or pg_stat_activity view. Others are optional. For details about their meanings, see section "Hint for Setting Slow SQL Control Rules." • **"select"**, **"insert"**, **"update"**, **"delete"**, and **"merge"**: keyword sequence for concurrency control, which is case insensitive and can be **NULL**. • **"resource"**: resource threshold for triggering instance-level concurrency control. The value is in the format of '{cpu-80, memory-70}', indicating the OS resource threshold for triggering instance-level concurrency control. The value can be **NULL**, indicating that concurrency control is performed regardless of the resource usage.

Example:

```
gaussdb=#  SELECT gs_add_workload_rule('sqlid', 'rule for one query', '', now(), '', 20, '{id=32413214}');
 gs_add_workload_rule
----------------------
                    1
(1 row)
gaussdb=#  CREATE database db1;
gaussdb=#  CREATE database db2;
gaussdb=#  SELECT gs_add_workload_rule('select', 'rule for select', '{db1, db2}', '', '', 100, '{tb1, tb2}');
 gs_add_workload_rule
----------------------
                    2
(1 row)
gaussdb=#  SELECT gs_add_workload_rule('resource', 'rule for resource', '{}', '', '', 20, '{cpu-80}');
 gs_add_workload_rule
----------------------
                    3
(1 row)
```

- gs_update_workload_rule(rule_id, rule_name, databases, start_time, end_time, max_workload, option_val)

  Description: To update an SQL statement concurrency control rule, users need to reset all parameters instead of only some parameters. Users must have the sysadmin permission.

  Parameters: For details, see **Table 7-165**.

  Return type: Boolean

  Table 7-165 gs_update_workload_rule parameters

Parameter	Type	Description	Value Range
rule_id	int8	ID of the concurrency control rule to be updated.	-
rule_name	name	Name of a concurrency control rule, which is used to search for the concurrency control rule.	Any character string or **NULL**.

Parameter	Type	Description	Value Range
databases	name[]	Array of database names for which the concurrency control rule takes effect. The value is case sensitive.	List of names of created databases. The value can be **NULL**, indicating that the configuration takes effect in all databases. Currently, the database list takes effect only when **rule_type** is set to a query type because a unique SQL ID is bound to a database and belongs to only one database. The concurrency control rules based on resource usage take effect for instances, that is, all databases.
start_time	timestamptz	Start time of a concurrency control rule.	The value can be **NULL**, indicating that it takes effect from now on.
end_time	timestamptz	End time of a concurrency control rule.	The value can be **NULL**, indicating that the rule is always effective.
max_workload	int8	Maximum number of concurrent requests set in a concurrency control rule.	-

Parameter	Type	Description	Value Range
option_val	text[]	Supplementary information about the concurrency control rule.	It matches **rule_type**. The matching relationship is as follows: • **"sqlid"**: specifies the unique ID of the SQL statement whose concurrency is to be controlled and slow SQL control rule. The format is '{id=1234, time_limit=100, max_execute_time=500, max_iops=1}', in which **id** indicates the unique SQL ID and is mandatory. You can obtain it from the dbe_perf.statement or pg_stat_activity view. Others are optional. For details about their meanings, see section "Hint for Setting Slow SQL Control Rules." • **"select"**, **"insert"**, **"update"**, **"delete"**, and **"merge"**: keyword sequence for concurrency control, which is case insensitive and can be **NULL**. • **"resource"**: resource threshold for triggering instance-level concurrency control. The value is in the format of '{cpu-80, memory-70}', indicating the OS resource threshold for triggering instance-level concurrency control. The value can be **NULL**, indicating that concurrency control is performed regardless of the resource usage.

Example:

```
gaussdb=#  CREATE database db1;
gaussdb=#  SELECT gs_update_workload_rule(2, 'rule for select 2', '{db1}', now(), '', 50, '{tb1}');
 gs_update_workload_rule
-------------------------
 t
(1 row)
```

- gs_delete_workload_rule(rule_id int8)

  Description: Deletes an SQL statement concurrency control rule. Users must have the sysadmin permission.

  Parameter: **rule_id** indicates the ID of the concurrency control rule to be updated. The type is int8.

  Return type: Boolean

  Example:

  ```
  gaussdb=#  SELECT gs_delete_workload_rule(3);
   gs_delete_workload_rule
  -------------------------
   t
  (1 row)
  ```

- gs_get_workload_rule_stat(rule_id)

  Description: Queries the number of times that SQL statements are blocked by SQL statement concurrency control rules. Users must have the sysadmin permission.

  Parameter: **rule_id** indicates the ID of the concurrency control rule to be queried. The type is int8. You can set **rule_id** to **–1**, indicating that all SQL statement concurrency control rules are queried.

  Table 3 Return value types

Name	Type	Description
rule_id	int8	ID of the SQL statement concurrency control rule.
validate_count	int8	Number of SQL statements intercepted by the SQL statement concurrency control rule.

  Example:

  ```
  gaussdb=#  SELECT * FROM gs_get_workload_rule_stat(1);
   rule_id | validate_count
  ---------+----------------
        1 |              0
  (1 row)
  gaussdb=#  SELECT * FROM gs_get_workload_rule_stat(-1);
   rule_id | validate_count
  ---------+----------------
        1 |              0
        2 |              0
  (2 rows)
  ```

7.6.48 Vector Calculation Interfaces and Functions

7.6.48.1 Vector Distance Calculation Interface

l2_distance

Function: Calculates the Euclidean distance between two vectors.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: float8

Code example:

```
gaussdb=# SELECT l2_distance(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT l2_distance('[1,2,3]',  '[5,-1,3.5]');
```

vector_l2_squared_distance

Function: Calculates the square of the Euclidean distance between two vectors.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: float8

Code example:

```
gaussdb=# SELECT vector_l2_squared_distance(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT vector_l2_squared_distance('[1,2,3]',  '[5,-1,3.5]');
```

cosine_distance

Function: Calculates the cosine distance between two vectors.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: float8

Code example:

```
gaussdb=# select cosine_distance(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# select cosine_distance('[1,2,3]', '[5,-1,3.5]');
```

vector_spherical_distance

Function: Calculates the spherical distance between two normalized vectors (represented by the radian system of the cosine angle).

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: float8

Code example:

```
gaussdb=# gaussdb=# select vector_spherical_distance('[1,0,0,0]', '[0,0,0,1]');
```


NOTE

If the input vector is not normalized, the correct calculation result cannot be obtained.

7.6.48.2 Vector Operation Function Interface

Functions implemented by the vector operation function include: vector size comparison, vector addition, vector subtraction, and vector bitwise multiplication.

inner_product

Function: Calculates the inner product of two vectors.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: float8

Code example:

```
gaussdb=# SELECT inner_product(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT inner_product('[1,2,3]', '[5,-1,3.5]');
```

vector_negative_inner_product

Function: Calculates the negative inner product of two vectors.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: float8

Code example:

```
gaussdb=# SELECT vector_negative_inner_product(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT vector_negative_inner_product('[1,2,3]', '[5,-1,3.5]');
```

vector_dims

Function: Returns the dimension value of the floatvector vector.

Input parameter type: floatvector

Output parameter type: int4

Code example:

```
gaussdb=# SELECT vector_dims(floatvector('[1,2,3]'));
gaussdb=# SELECT vector_dims('[1,2,3]');
```

vector_norm

Function: Returns the L2 norm of a vector.

Input parameter type: floatvector

Output parameter type: float8

Code example:


```
gaussdb=# SELECT vector_norm(floatvector('[1,2,3]'));
gaussdb=# SELECT vector_norm('[1,2,3]');
```

vector_add

Function: Calculates the sum of two vectors.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: floatvector

Code example:

```
gaussdb=# SELECT vector_add(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT vector_add('[1,2,3]', '[5,-1,3.5]');
```

vector_sub

Function: Calculates the subtraction between two vectors.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: floatvector

Code example:

```
gaussdb=# SELECT vector_sub(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT vector_sub('[1,2,3]', '[5,-1,3.5]');
```

vector_lt

Function: Compares the vector size and checks whether vector 1 is less than vector 2.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: Boolean

Code example:

```
gaussdb=# SELECT vector_lt(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT vector_lt('[1,2,3]', '[5,-1,3.5]');
```

vector_le

Function: Compares the vector size and checks whether vector 1 is less than or equal to vector 2.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: Boolean

Code example:


```
gaussdb=# SELECT vector_le(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT vector_le('[1,2,3]', '[5,-1,3.5]');
```

vector_eq

Function: Compares whether vectors are equal.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: Boolean

Code example:

```
gaussdb=# SELECT vector_eq(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT vector_eq('[1,2,3]', '[5,-1,3.5]');
```

vector_ne

Function: Compares whether vectors are unequal.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: Boolean

Code example:

```
gaussdb=# SELECT vector_ne(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT vector_ne('[1,2,3]', '[5,-1,3.5]');
```

vector_ge

Function: Compares the vector size and checks whether vector 1 is greater than or equal to vector 2.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: Boolean

Code example:

```
gaussdb=# SELECT vector_ge(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT vector_ge('[1,2,3]', '[5,-1,3.5]');
```

vector_gt

Function: Compares the vector size and checks whether vector 1 is greater than vector 2.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: Boolean

Code example:


```
gaussdb=# SELECT vector_gt(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT vector_gt('[1,2,3]', '[5,-1,3.5]');
```

vector_cmp

Function: Compares vector sizes.

Type of input parameter 1: floatvector

Type of input parameter 2: floatvector

Output parameter type: int4

Code example:

```
gaussdb=# SELECT vector_cmp(floatvector('[1,2,3]'), floatvector('[5,-1,3.5]'));
gaussdb=# SELECT vector_cmp('[1,2,3]', '[5,-1,3.5]');
```

vector_accum

Function: Returns the vector accumulation.

Type of input parameter 1: anyarray

Type of input parameter 2: floatvector

Output parameter type: anyarray

Code example:

```
-- This is a system function and is not recommended. If this function is required, the array element type must be float8.
gaussdb=# SELECT vector_accum(array[cast(3 as float8),1,2,3],  floatvector('[5,-1,3.5]'));
```

vector_combine

Function: Combines vectors.

Type of input parameter 1: anyarray

Type of input parameter 2: anyarray

Output parameter type: anyarray

Code example:

```
-- This is a system function and is not recommended. If this function is required, the array element type must be float8.
gaussdb=# SELECT vector_combine(array[cast(1 as float8),2,3], array[cast(1 as float8),2,3]);
```

vector_avg

Function: Averages vectors.

Input parameter type: anyarray

Output parameter type: floatvector

Code example:

```
-- This is a system function and is not recommended. If this function is required, the array element type must be float8.
gaussdb=# SELECT vector_avg(array[cast(1 as float8),2,3]);
```


bool_vector_dims

Function: Returns the dimension value of the boolvector vector.

Input parameter type: boolvector

Output parameter type: int4

Code example:

```
gaussdb=# SELECT bool_vector_dims(boolvector('[1,1,1]'));
```

bool_vector_eq

Function: Compares whether Boolean vectors are equal.

Type of input parameter 1: boolvector

Type of input parameter 2: boolvector

Output parameter type: Boolean

Code example:

```
gaussdb=# SELECT bool_vector_eq(boolvector('[1,1,1]'), boolvector('[1,1,1]'));
gaussdb=# SELECT bool_vector_eq('[1,1,1]', '[1,1,1]');
```

7.6.48.3 Vector Functions and Operators

floatvector supports data conversion between the vector type and array type. In addition, strings in specific formats can be converted to the vector type.

array<->floatvector: During data type conversion, the vector data type can be freely converted to the corresponding array type. The member data type of the floatvector vector is floating point.

string ->floatvector: You need to pay attention to the character string format. You need to contain arrays using square brackets ([]) or braces ({}) and separate elements using commas (,).

floatvector

Function: Converts array data into vector data.

Scenario 1:

Input parameter type: anyarray

Output parameter type: floatvector

Code example:

```
gaussdb=# SELECT floatvector(ARRAY[1,2,9.3]);
```

Scenario 2:

Type of input parameter 1: floatvector

Type of input parameter 2: integer

Output parameter type: floatvector

Code example:

```
gaussdb=# SELECT floatvector(floatvector('[1,2,9.3]'),3);
```

vector_to_array

Function: Converts vector data into array data.

Input parameter type: floatvector

Output parameter type: real[]

Code example:

```
gaussdb=# SELECT vector_to_array(floatvector('[1,2,3]'));
```

text_to_vector

Function: Converts character data into vector data.

Input parameter type: cstring

Output parameter type: floatvector

Code example:

```
gaussdb=# SELECT text_to_vector('[1,2,9.3]');
```

vector_in

Function: Specifies the floatvector input conversion function (text format).

Type of input parameter 1: cstring

Type of input parameter 2: OID

Type of input parameter 3: int4

Output parameter type: floatvector

Code example:

```
gaussdb=# SELECT vector_in('[1,2,3]',701,3);
```

vector_out

Function: Specifies the floatvector output conversion function (text format).

Input parameter type: floatvector

Output parameter type: cstring

Code example:

```
gaussdb=# SELECT vector_out(floatvector('[1,2,3]'));
```

vector_send

Function: Specifies the floatvector input conversion function (binary format).

Input parameter type: floatvector

Output parameter type: bytea

Code example:

```
gaussdb=# SELECT vector_send(floatvector('[1,2,3]'));
```

vector_recv

Function: Specifies the floatvector output conversion function (binary format).

Type of input parameter 1: integer

Type of input parameter 2: OID

Type of input parameter 3: int4

Output parameter type: floatvector

Code example:

```
-- This is a system function and cannot be called by SQL statements.
```

vector_typmod_in

Function: Specifies the floatvector input type modifier function.

Input parameter type: cstring[]

Output parameter type: integer

Code example:

```
gaussdb=# SELECT vector_typmod_in( '{1}' );
```

NOTE

- Vector data type members support only single precision.
- Only the same dimension is supported for vector calculation. If the dimensions are different, an error is reported.
- floatvector supports vector addition and subtraction. The dot product operation is performed by the inner_product function.
- When creating a table, you must specify dimensions for each vector type. If the dimensions of the inserted data are different from the specified dimensions, the database reports an error.
- If data has been inserted into the data table, the data in the table must comply with the dimension settings when the vector dimensions are changed. Otherwise, the database reports an error. If a vector index has been created for a vector attribute, the data type of the attribute cannot be switched to another data type, and the database reports an error.

boolvector

Scenario 1:

Function: Converts array data into vector data.

Input parameter type: anyarray

Output parameter type: boolvector

Code example:


```
gaussdb=# SELECT boolvector(ARRAY[1,0,1]);
```

Scenario 2:

Function: Converts to boolvector and detects dimensions.

Type of input parameter 1: boolvector

Type of input parameter 2: integer

Output parameter type: boolvector

Code example:

```
gaussdb=# SELECT boolvector(boolvector('[1,0,1]'),3);
```

boolvector_to_array

Function: Converts vector data into array data.

Input parameter type: boolvector

Output parameter type: anyarray

Code example:

```
gaussdb=# SELECT boolvector_to_array(boolvector('[1,0,1]'));
```

text_to_boolvector

Function: Converts character data into vector data.

Input parameter type: cstring

Output parameter type: boolvector

Code example:

```
gaussdb=# SELECT text_to_boolvector('[1,1,1]');
```

bool_vector_in

Function: Specifies the boolvector input conversion function (text format).

Type of input parameter 1: cstring

Type of input parameter 2: OID

Type of input parameter 3: int4

Output parameter type: boolvector

Code example:

```
gaussdb=# SELECT bool_vector_in('[1,1,1]',701,3);
```

bool_vector_out

Function: Specifies the boolvector output conversion function (text format).

Input parameter type: boolvector

Output parameter type: cstring

Code example:

```
gaussdb=# SELECT bool_vector_out(boolvector('[1,1,1]'));
```

bool_vector_send

Function: Specifies the boolvector input conversion function (binary format).

Input parameter type: boolvector

Output parameter type: bytea

Code example:

```
gaussdb=# SELECT bool_vector_send(boolvector('[1,1,1]'));
```

bool_vector_recv

Function: Specifies the boolvector output conversion function (binary format).

Type of input parameter 1: internal

Type of input parameter 2: OID

Type of input parameter 3: int4

Output parameter type: boolvector

Code example:

```
-- This is a system function and cannot be called by SQL statements.
```

bool_vector_typmod_in

Function: Specifies the boolvector input type modifier function.

Input parameter type: cstring[]

Output parameter type: integer

Code example:

```
gaussdb=# SELECT bool_vector_typmod_in( '{1}' );
```

NOTE

- The vector data type does not support null, nan, or inf as elements. If a vector contains **NULL** values, the database reports an error.
- When a vector data type is inserted, **NULL** cannot be used as the inserted value. When a **NULL** value is inserted as the vector data, the database reports an error.
- Elements of the boolvector type can express Boolean data in **t(T)**/**f(F)**, **y(Y)**/**n(N)**, **1**/**0**, **true**/**false**, **yes**/**no**, or **on**/**off** mode.
- The boolvector type applies only to the equal-to operation and does not support comparison operations.

gs_vector_index_options

Function: Displays the hyperparameter values of related vector indexes.

Input parameter type: text

Output parameter type: text

Code example:

```
-- Create a table.
gaussdb=# CREATE TABLE t1 (id int unique,repr floatvector(960)) with (storage_type=astore);
Insert data.
gaussdb=# copy t1 from '/data/gist1w.txt' delimiter '^';
Create an index.
gaussdb=# CREATE INDEX test1v on t1 using gsdiskann (repr l2) with
(pq_nseg=120,pq_nclus=64,queue_size=120,num_parallels=30,enable_pq=true,using_clustering_for_parallel=f
alse);

gaussdb=# SELECT gs_vector_index_options('test1v');
```

Operators for vector calculation focus on measuring similarity between vectors and binary operators of vectors.

<->

Function: Calculates the Euclidean distance (L2) between two vectors (floatvector).

Left parameter type: floatvector

Right parameter type: floatvector

Return type: double precision

Code example:

```
gaussdb=# SELECT floatvector('[1,1,3,2]') <->  floatvector('[1,1,3,2]');
gaussdb=# SELECT '[1,2,3,2]'<-> floatvector('[1,1,3,2]');
```

<+>

Function: Calculates the cosine distance between two vectors (floatvector).

Left parameter type: floatvector

Right parameter type: floatvector

Return type: double precision

Code example:

```
gaussdb=# SELECT floatvector('[1,1,3,2]') <+>  floatvector('[1,1,3,2]');
gaussdb=# SELECT '[1,2,3,2]'<+> floatvector('[1,1,3,2]');
```

<#>

Function: Calculates the Hamming distance between two vectors (floatvector).

Left parameter type: boolvector

Right parameter type: boolvector

Return type: double precision

Code example:

```
gaussdb=# SELECT boolvector('[1,0,1,0]') <#>  boolvector('[1,1,1,0]');
gaussdb=# SELECT '[1,0,1,0]'<#> boolvector('[1,1,1,0]');
```


+

Function: Calculates the bitwise addition of two vectors with the same dimension.

Left parameter type: floatvector

Right parameter type: floatvector

Return type: floatvector

Code example:

```
gaussdb=# SELECT floatvector('[1,1,3,2]') + floatvector('[1,1,3,2]');
gaussdb=# SELECT '[1,2,3,2]'+ floatvector('[1,1,3,2]');
```

-

Function: Calculates the bitwise subtraction of two vectors with the same dimension.

Left parameter type: floatvector

Right parameter type: floatvector

Return type: floatvector

Code example:

```
gaussdb=# SELECT floatvector('[1,1,3,2]') + floatvector('[1,1,3,2]');
gaussdb=# SELECT '[1,2,3,2]'+ floatvector('[1,1,3,2]');
```

<

Function: Compares the lexicographical order of two vectors with the same dimension.

Left parameter type: floatvector

Right parameter type: floatvector

Return type: Boolean

Code example:

```
gaussdb=# SELECT floatvector('[1,1,3,2]') < floatvector('[1,1,3,2]');
gaussdb=# SELECT '[1,2,3,2]'< floatvector('[1,1,3,2]');
```

<=

Function: Compares the lexicographical order of two vectors with the same dimension.

Left parameter type: floatvector

Right parameter type: floatvector

Return type: Boolean

Code example:

```
gaussdb=# SELECT floatvector('[1,1,3,2]') <= floatvector('[1,1,3,2]');
gaussdb=# SELECT '[1,2,3,2]'<= floatvector('[1,1,3,2]');
```


>

Function: Compares the lexicographical order of two vectors with the same dimension.

Left parameter type: floatvector

Right parameter type: floatvector

Return type: Boolean

Code example:

```
gaussdb=# SELECT floatvector('[1,1,3,2]') > floatvector('[1,1,3,2]');
gaussdb=# SELECT '[1,2,3,2]'> floatvector('[1,1,3,2]');
```

>=

Function: Compares the lexicographical order of two vectors with the same dimension.

Left parameter type: floatvector

Right parameter type: floatvector

Return type: Boolean

Code example:

```
gaussdb=# SELECT floatvector('[1,1,3,2]') >= floatvector('[1,1,3,2]');
gaussdb=# SELECT '[1,2,3,2]'>= floatvector('[1,1,3,2]');
```

=

Scenario 1:

Function: Determines whether two vectors with the same dimension are equal.

Left parameter type: floatvector

Right parameter type: floatvector

Return type: Boolean

Code example:

```
gaussdb=# SELECT floatvector('[1,1,3,2]') = floatvector('[1,1,3,2]');
gaussdb=# SELECT '[1,2,3,2]'= floatvector('[1,1,3,2]');
```

Scenario 2:

Function: Determines whether two Boolean vectors with the same dimension are consistent.

Left parameter type: boolvector

Right parameter type: boolvector

Return type: Boolean

Code example:

```
gaussdb=# SELECT boolvector('[1,0,1,0]') =  boolvector('[1,1,1,0]');
gaussdb=# SELECT '[1,0,1,0]' =  boolvector('[1,1,1,0]');
```


<>

Function: Determines whether two vectors with the same dimension are unequal.

Left parameter type: floatvector

Right parameter type: floatvector

Return type: Boolean

Code example:

```
gaussdb=# SELECT floatvector('[1,1,3,2]') <> floatvector('[1,1,3,2]');
gaussdb=# SELECT '[1,2,3,2]'<> floatvector('[1,1,3,2]');
```

NOTE

If the vector element value is large, the intermediate calculation result may overflow the single-precision range during distance calculation. As a result, the distance result is NAN or INF.

7.6.49 Deprecated Functions

The following functions in GaussDB have been discarded in the latest version:

- gs_wlm_get_session_info
- gs_wlm_get_user_session_info
- pgxc_get_csn
- pgxc_get_stat_dirty_tables
- pgxc_get_thread_wait_status
- pgxc_gtm_snapshot_status
- pgxc_is_committed
- pgxc_lock_for_backup
- pgxc_lock_for_sp_database
- pgxc_lock_for_transfer
- pgxc_log_comm_status
- pgxc_max_datanode_size
- pgxc_node_str
- pgxc_pool_check
- pgxc_pool_connection_status
- pgxc_pool_reload
- pgxc_prepared_xact
- pgxc_snapshot_status
- pgxc_stat_dirty_tables
- pgxc_unlock_for_sp_database
- pgxc_unlock_for_transfer
- pgxc_version
- array_extend
- prepare_statement_status

- remote_rto_stat
- dbe_perf.global_slow_query_info
- dbe_perf.global_slow_query_info_bytime
- dbe_perf.global_slow_query_history
- pg_stat_get_pooler_status
- pg_stat_get_wlm_node_resource_info
- pg_stat_get_wlm_session_info_internal
- DBE_PERF.get_wlm_controlgroup_ng_config()
- DBE_PERF.get_wlm_user_resource_runtime()
- global_space_shrink
- pg_pool_validate
- gs_stat_ustore
- table_skewness(text)
- table_skewness(text, text, text)
- local_segment_space_info
- pg_stat_segment_extent_usage
- GS_ALL_NODEGROUP_CONTROL_GROUP_INFO(text)
- create_wlm_operator_info(int flag)
- create_wlm_session_info(int flag)
- pg_stat_get_wlm_session_info(int flag)
- gs_wlm_get_resource_pool_info(int)
- gs_wlm_get_all_user_resource_info()
- gs_wlm_get_workload_records()
- gs_wlm_persistent_user_resource_info()
- gs_wlm_session_respool(bigint)
- gs_total_nodegroup_memory_detail
- gs_wlm_user_resource_info(name text)
- create_wlm_instance_statistics_info
- pg_stat_get_session_wlmstat
- pg_stat_get_wlm_ec_operator_info
- pg_stat_get_wlm_instance_info
- pg_stat_get_wlm_instance_info_with_cleanup
- pg_stat_get_wlm_statistics
- pg_stat_get_wlm_operator_info
- pg_stat_get_wlm_realtime_ec_operator_info
- pg_stat_get_wlm_realtime_operator_info
- pg_stat_get_wlm_realtime_session_info
- get_node_modulo
- check_murmurhash_route_node
- gs_redis_set_bucketxid(bigint)

7.7 Expressions

7.7.1 Simple Expressions

Logical Expressions

Logical Operators lists the operators and calculation rules of logical expressions.

Comparative Expressions

Comparison Operators lists the common comparative operators.

In addition to comparative operators, you can also use the following sentence structure:

- BETWEEN operators:

  a BETWEEN x AND y is equivalent to **a >= x AND a <= y**.

  a NOT BETWEEN x AND y is equivalent to **a < x OR a > y**.
- To check whether a value is null, use:

  expression IS NULL

  expression IS NOT NULL

  or an equivalent (non-standard) sentence structure:

  expression ISNULL

  expression NOTNULL

  NOTICE

  - Do not write **expression=NULL** or **expression<>(!=)NULL**, because **NULL** represents an unknown value and these expressions cannot determine whether two unknown values are equal.
  - Only the comparative expressions IS NULL and IS NOT NULL support data of XML type.
- is distinct from/is not distinct from
  - is distinct from

    If the data types and values of A and B are different, the value is **true**.

    If the data types and values of A and B are the same, the value is **false**.

    Empty values are considered the same.
  - is not distinct from

    If the data types and values of A and B are different, the value is **false**.

    If the data types and values of A and B are the same, the value is **true**.

    Empty values are considered the same.
- <=> NULL-safe equal operator

The comparison of NULL values is added on the basis of the comparison of '='. If neither the left nor right value of the operator is NULL, the result is the same as that of '='.

If the data types and values of A and B are different, the value is **false**.

If the data types and values of A and B are the same, the value is **true**.

Empty values are considered the same.

NOTE

- The usage of the <=> operator is the same as that of IS NOT DISTINCT FROM.
- This operator is valid only when the database is B-compatible (that is, **sql_compatibility** is set to **'B'**). Other types do not support this operator.

Examples

```
gaussdb=# SELECT 2 BETWEEN 1 AND 3 AS RESULT;
 result
----------
 t
(1 row)

gaussdb=# SELECT 2 >= 1 AND 2 <= 3 AS RESULT;
 result
----------
 t
(1 row)

gaussdb=# SELECT 2 NOT BETWEEN 1 AND 3 AS RESULT;
 result
----------
 f
(1 row)

gaussdb=# SELECT 2 < 1 OR 2 > 3 AS RESULT;
 result
----------
 f
(1 row)

gaussdb=# SELECT 2+2 IS NULL AS RESULT;
 result
----------
 f
(1 row)

gaussdb=# SELECT 2+2 IS NOT NULL AS RESULT;
 result
----------
 t
(1 row)

gaussdb=# SELECT 2+2 ISNULL AS RESULT;
 result
----------
 f
(1 row)

gaussdb=# SELECT 2+2 NOTNULL AS RESULT;
 result
----------
 t
(1 row)

gaussdb=# SELECT 2+2 IS DISTINCT FROM NULL AS RESULT;
 result
```



```
----------
 t
(1 row)

gaussdb=# SELECT 2+2 IS NOT DISTINCT FROM NULL AS RESULT;
 result
----------
 f
(1 row)

gaussdb=# SELECT 1 <=> 1 AS RESULT;
 result
--------
 t
(1 row)

gaussdb=# SELECT NULL <=> 1 AS RESULT;
 result
--------
 f
(1 row)

gaussdb=# SELECT NULL <=> NULL AS RESULT;
 result
--------
 t
(1 row)
```

7.7.2 Condition Expressions

Data that meets the requirements specified by conditional expressions are filtered during SQL statement execution.

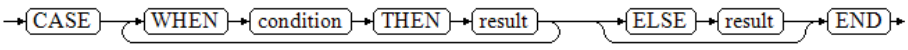
Conditional expressions include the following types:

- CASE

  CASE expressions are similar to the **CASE** statements in other coding languages.

  Figure 7-1 shows the syntax of a CASE expression.

  Figure 7-1 case::=

  A CASE clause can be used in a valid expression. The condition is an expression that returns a value of Boolean type.

  - If the result is true, the result of the CASE expression is the required result.
  - If the result is false, the following WHEN or ELSE clauses are processed in the same way.
  - If every WHEN condition is not true, the result of the expression is the result of the ELSE clause. If the ELSE clause is omitted and has no match condition, the result is NULL.
  - Operations on XML data are supported.

  Example:

  ```
  gaussdb=# CREATE TABLE case_when_t1(CW_COL1 INT);

  gaussdb=# INSERT INTO case_when_t1 VALUES (1), (2), (3);
  ```

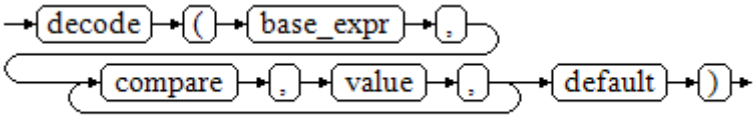


```
gaussdb=# SELECT * FROM case_when_t1;
cw_col1
---------
 1
 2
 3
(3 rows)

gaussdb=# SELECT CW_COL1, CASE WHEN CW_COL1=1 THEN 'one' WHEN CW_COL1=2 THEN 'two'
ELSE 'other' END FROM case_when_t1 ORDER BY 1;
 cw_col1 | case
---------+-------
       1 | one
       2 | two
       3 | other
(3 rows)

gaussdb=# DROP TABLE case_when_t1;
```

- DECODE

  Figure 7-2 shows the syntax of DECODE.

  Figure 7-2 decode::=

  Compare each following compare(n) with base_expr. **value(n)** is returned if a compare(n) matches the base_expr expression. If base_expr does not match each compare(n), the default value is returned.

  Operations on XML data are supported.

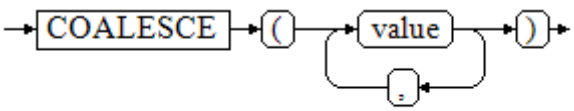
  Conditional Expression Functions describes the examples.

  ```
  gaussdb=# SELECT DECODE('A','A',1,'B',2,0);
   case
  ------
     1
  (1 row)
  ```

- COALESCE

  Figure 7-3 shows the syntax of COALESCE.

  Figure 7-3 coalesce::=

  COALESCE returns its first not-NULL value. If all the parameters are **NULL**, **NULL** is returned. This value is replaced by the default value when data is displayed. Like a CASE expression, COALESCE only evaluates the parameters that are needed to determine the result. That is, parameters to the right of the first non-null parameter are not evaluated.

  Operations on XML data are supported.

Example:

```
gaussdb=# CREATE TABLE c_tabl(description varchar(10), short_description varchar(10), last_value
varchar(10)) ;

gaussdb=# INSERT INTO c_tabl VALUES('abc', 'efg', '123');
gaussdb=# INSERT INTO c_tabl VALUES(NULL, 'efg', '123');

gaussdb=# INSERT INTO c_tabl VALUES(NULL, NULL, '123');

gaussdb=# SELECT description, short_description, last_value, COALESCE(description, short_description,
last_value) FROM c_tabl ORDER BY 1, 2, 3, 4;
 description | short_description | last_value | coalesce
-------------+-------------------+------------+----------
 abc         | efg               | 123        | abc
             | efg               | 123        | efg
             |                   | 123        | 123
(3 rows)

gaussdb=# DROP TABLE c_tabl;
```

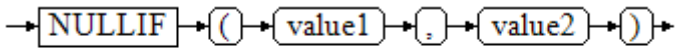
If **description** is not **NULL**, the value of **description** is returned. Otherwise, **short_description** is calculated. If **short_description** is not **NULL**, the value of **short_description** is returned. Otherwise, **last_value** is calculated. If **last_value** is not **NULL**, the value of **last_value** is returned. Otherwise, **none** is returned.

```
gaussdb=# SELECT COALESCE(NULL,'Hello World');
   coalesce
---------------
 Hello World
(1 row)
```

- NULLIF

  Figure 7-4 shows the syntax of NULLIF.

  Figure 7-4 nullif::=

  Only if **value1** is equal to **value2** can **NULLIF** return the **NULL** value. Otherwise, **value1** is returned. Operations on XML data are supported.

  Example:

```
gaussdb=# CREATE TABLE null_if_t1 (
    NI_VALUE1 VARCHAR(10),
    NI_VALUE2 VARCHAR(10)
);

gaussdb=# INSERT INTO null_if_t1 VALUES('abc', 'abc');
gaussdb=# INSERT INTO null_if_t1 VALUES('abc', 'efg');

gaussdb=# SELECT NI_VALUE1, NI_VALUE2, NULLIF(NI_VALUE1, NI_VALUE2) FROM null_if_t1 ORDER
BY 1, 2, 3;

 ni_value1 | ni_value2 | nullif
-----------+-----------+--------
 abc       | abc       |
 abc       | efg       | abc
(2 rows)
gaussdb=# DROP TABLE null_if_t1;
```

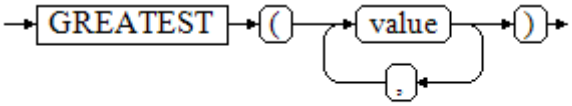
  If the value of **value1** is equal to that of **value2**, **NULL** is returned. Otherwise, the value of **value1** is returned.


```
gaussdb=# SELECT NULLIF('Hello','Hello World');
 nullif
--------
 Hello
(1 row)
```

- GREATEST (maximum value) and LEAST (minimum value)

  Figure 7-5 shows the syntax of GREATEST.

  Figure 7-5 greatest::=

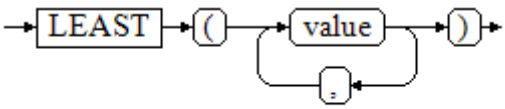
  

  You can select the maximum value from any numerical expression list. Operations on XML data are supported.

  ```
  gaussdb=# SELECT greatest(9000,155555,2.01);
   greatest
  ----------
    155555
  (1 row)
  ```

  Figure 7-6 shows the syntax of LEAST.

  Figure 7-6 least::=

  You can select the minimum value from any numerical expression list.

  Each of the preceding numeric expressions can be converted into a common data type, which will be the data type of the result.

  The NULL values in the list will be ignored. The result is **NULL** only if the results of all expressions are **NULL**.

  Operations on XML data are supported.

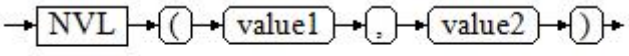
  ```
  gaussdb=# SELECT least(9000,2);
   least
  -------
      2
  (1 row)
  ```

  Conditional Expression Functions describes the examples.

- NVL

  Figure 7-7 shows the syntax of NVL.

  Figure 7-7 nvl::=

If the value of **value1** is **NULL**, **value2** is returned. Otherwise, **value1** is returned. Operations on XML data are supported.

Example:

```
gaussdb=# SELECT nvl(null,1);
nvl
-----
 1
(1 row)
gaussdb=# SELECT nvl('Hello World',1);
      nvl
---------------
 Hello World
(1 row)
```

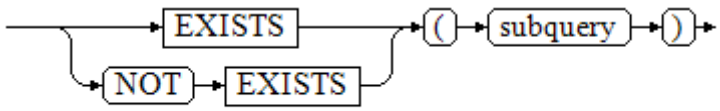
7.7.3 Subquery Expressions

Subquery expressions include the following types:

- EXISTS/NOT EXISTS

  Figure 7-8 shows the syntax of EXISTS/NOT EXISTS.

  Figure 7-8 EXISTS/NOT EXISTS::=

  The parameter of an **EXISTS** expression is an arbitrary **SELECT** statement, that is, subquery. The subquery is evaluated to determine whether it returns any rows. If it returns at least one row, the result of EXISTS is true. If the subquery returns no rows, the result of EXISTS is false.

  The subquery will generally only be executed long enough to determine whether at least one row is returned, not all the way to completion.

  Operations on XML data are not supported.

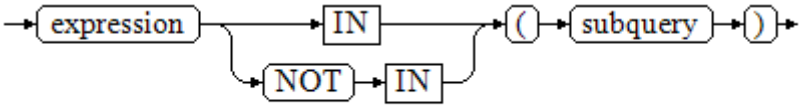
  Example:

  ```
  gaussdb=# CREATE TABLE exists_t1(a int, b int);
  gaussdb=# INSERT INTO exists_t1 VALUES(1, 2),(2, 3),(3, 4),(4, 5);

  gaussdb=# CREATE TABLE exists_t2(a int, c int);
  gaussdb=# INSERT INTO exists_t2 VALUES(3, 4),(4, 5),(5, 6),(6, 7);

  gaussdb=# SELECT * FROM exists_t1 t1 WHERE EXISTS (SELECT * FROM exists_t2 t2 WHERE t2.a =
  t1.a);
   a | b
  ---+---
   3 | 4
   4 | 5
  (2 rows)

  gaussdb=# DROP TABLE exists_t1, exists_t2;
  ```

- IN/NOT IN

  Figure 7-9 shows the syntax of IN/NOT IN.

Figure 7-9 IN/NOT IN::=



The right-hand side is a parenthesized subquery, which must return exactly one column. The left-hand expression is evaluated and compared to each row of the subquery result. The result of IN is true if any equal subquery row is found. The result is false if no equal row is found (including the case where the subquery returns no rows).

This is in accordance with SQL normal rules for Boolean combinations of null values. If the columns corresponding to two rows equal and are not empty, the two rows are equal to each other. If any columns corresponding to the two rows do not equal and are not empty, the two rows are not equal to each other. Otherwise, the result is **NULL**. If the results in each row are either unequal or NULL, with at least one NULL, the result of IN is null.

Operations on XML data are not supported.

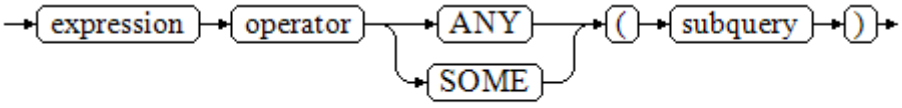
Example:

```
gaussdb=# CREATE TABLE in_t1(a int, b int);
gaussdb=# INSERT INTO in_t1 VALUES(1, 2),(2, 3),(3, 4),(4, 5);

gaussdb=# CREATE TABLE in_t2(a int, c int);
gaussdb=# INSERT INTO in_t2 VALUES(3, 4),(4, 5),(5, 6),(6, 7);

gaussdb=# SELECT * FROM in_t1 t1 WHERE t1.a IN (SELECT t2.a FROM in_t2 t2);
 a | b
---+---
 3 | 4
 4 | 5
(2 rows)

gaussdb=# DROP TABLE in_t1, in_t2;
```

- ANY/SOME

  Figure 7-10 shows the syntax of ANY/SOME.

  Figure 7-10 any/some::=



  The right-hand side is a parenthesized subquery, which must return exactly one column. The left-hand expression is evaluated and compared to each row of the subquery result using the given operator, which must yield a Boolean result. The result of ANY is true if any true result is obtained. The result is false if no true result is found (including the case where the subquery returns no rows). **SOME** is a synonym of **ANY**. IN can be equivalently replaced with ANY.

  Operations on XML data are not supported.

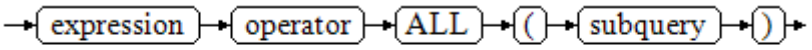
Example:

```
gaussdb=# CREATE TABLE any_t1(a int, b int);
gaussdb=# INSERT INTO any_t1 VALUES(1, 2),(2, 3),(3, 4),(4, 5);

gaussdb=# CREATE TABLE any_t2(a int, c int);
gaussdb=# INSERT INTO any_t2 VALUES(3, 4),(4, 5),(5, 6),(6, 7);

gaussdb=# SELECT * FROM any_t1 t1 WHERE t1.a < ANY(SELECT t2.a FROM any_t2 t2 where t2.a = 3
or t2.a = 4);
 a | b
---+---
 1 | 2
 2 | 3
 3 | 4
(3 rows)

gaussdb=# DROP TABLE any_t1, any_t2;
```

- ALL

  Figure 7-11 shows the syntax of ALL.

  Figure 7-11 all::=

  The right-hand side is a parenthesized subquery, which must return exactly one column. The left-hand expression is evaluated and compared to each row of the subquery result using the given operator, which must yield a Boolean result. The result of ALL is true if all values are true (including the case where the subquery returns no rows). The result is false if any false result is found.

  Operations on XML data are not supported.

  Example:

```
gaussdb=# CREATE TABLE all_t1(a int, b int);
gaussdb=# INSERT INTO all_t1 VALUES(1, 2),(2, 3),(3, 4),(4, 5);

gaussdb=# CREATE TABLE all_t2(a int, c int);
gaussdb=# INSERT INTO all_t2 VALUES(3, 4),(4, 5),(5, 6),(6, 7);

gaussdb=# SELECT * FROM all_t1 t1 WHERE t1.a < ALL(SELECT t2.a FROM all_t2 t2 where t2.a = 3 or
t2.a = 4);
 a | b
---+---
 1 | 2
 2 | 3
(2 rows)

gaussdb=# DROP TABLE all_t1, all_t2;
```

7.7.4 Array Expressions

IN

expression **IN** *(value [, ...])*

The parentheses on the right contain an expression list. The expression result on the left is compared with the content in the expression list. If the content in the list meets the expression result on the left, the result of IN is true. If no result meets the requirements, the result of IN is false.

Example:

```
gaussdb=# SELECT 8000+500 IN (10000, 9000) AS RESULT;
 result
----------
 f
(1 row)
```

NOTE

- If the expression result is null or the expression list does not meet the expression conditions and at least one empty value is returned for the expression list on the right, the result of IN is null rather than false. This method is consistent with the Boolean rules used when SQL statements return empty values.
- Operations on XML data are not supported.

NOT IN

expression **NOT IN** *(value [, ...])*

The parentheses on the right contain an expression list. The expression result on the left is compared with the content in the expression list. If the content in the list does not meet the expression result on the left, the result of NOT IN is true. If any content meets the expression result, the result of NOT IN is false.

Example:

```
gaussdb=# SELECT 8000+500 NOT IN (10000, 9000) AS RESULT;
 result
----------
 t
(1 row)
```

NOTE

- If the query statement result is null or the expression list does not meet the expression conditions and at least one empty value is returned for the expression list on the right, the result of NOT IN is null rather than false. This method is consistent with the Boolean rules used when SQL statements return empty values.
- In all situations, X NOT IN Y equals to NOT(X IN Y).
- Operations on XML data are not supported.

ANY/SOME (array)

expression operator **ANY** *(array expression)*

expression operator **SOME** *(array expression)*

The right side is a parenthesized expression, which must yield an array value. The result of the expression on the left uses operators to compute and compare the results in each row of the array expression. The comparison result must be a Boolean value.

Example:

```
gaussdb=# SELECT 8000+500 < SOME (array[10000,9000]) AS RESULT;
 result
----------
 t
(1 row)
gaussdb=# SELECT 8000+500 < ANY (array[10000,9000]) AS RESULT;
 result
```



```
----------
 t
(1 row)
```

NOTE

- If at least one comparison result is true, the result of ANY is true.
- If no comparison result is true, the result of ANY is false.
- If no comparison result is true and the array expression generates at least one null value, the value of **ANY** is **NULL** rather than **false**. This method is consistent with the Boolean rules used when SQL statements return empty values.
- SOME is a synonym of ANY.
- Operations on XML data are not supported.

ALL (array)

expression operator **ALL** *(array expression)*

The right-hand side is a parenthesized expression, which must yield an array value. The result of the expression on the left uses operators to compute and compare the results in each row of the array expression. The comparison result must be a Boolean value.

- The result of ALL is true if all comparisons yield true (including the case where the array has zero elements).
- If one or more comparison results are false, the result of ALL is false.
- If the array expression yields a null array, the result of **ALL** will be null. If the expression on the left yields **NULL**, the result of ALL is generally **NULL** (though a non-strict comparison operator could possibly yield a different result). If the array on the right contains any null elements and no false comparison result is found, the result of **ALL** is **NULL**, not true (again, assuming a strict comparison operator). This method is consistent with the Boolean rules used when SQL statements return empty values.
- Operations on XML data are not supported.

```
gaussdb=# SELECT 8000+500 < ALL (array[10000,9000]) AS RESULT;
 result
----------
 t
(1 row)
```

7.7.5 Row Expressions

The syntax is as follows:

```
row_constructor operator row_constructor
```

Both sides of the row expression are row constructors. The values of both rows must have the same number of fields and they are compared with each other. The row comparison allows operators including =, <>, <, <=, and >= or a similar operator.

For operators <, <=, >, and > =, the columns in rows are compared from left to right until a pair of columns that are not equal or are empty are detected. If the pair of columns contains at least one null value, the comparison result is null. Otherwise, the comparison result of this pair of columns is the final result. If no

unequal or empty column is found, the values in the two rows are equal. The final result is determined based on the operator meaning.

Operations on XML data are not supported.

Example:

```
gaussdb=# SELECT ROW(1,2,NULL) < ROW(1,3,0) AS RESULT;
 result
----------
 t
(1 row)

gaussdb=# SELECT (4,5,6) > (3,2,1) AS result;
 result
--------
 t
(1 row)

gaussdb=# SELECT (4,1,1) > (3,2,1) AS result;
 result
--------
 t
(1 row)

gaussdb=# SELECT ('test','data') > ('data','data') AS result;
 result
--------
 t
(1 row)

gaussdb=# SELECT (4,1,1) > (3,2,null) AS result;
 result
--------
 t
(1 row)

gaussdb=# SELECT (null,1,1) > (3,2,1) AS result;
 result
--------

(1 row)

gaussdb=# SELECT (null,5,6) > (null,5,6) AS result;
 result
--------

(1 row)

gaussdb=# SELECT (4,5,6) > (4,5,6) AS result;
 result
--------
 f
(1 row)

gaussdb=# SELECT (2,2,5) >= (2,2,3) AS result;
 result
--------
 t
(1 row)

gaussdb=# SELECT (2,2,1) <= (2,2,3) AS result;
 result
--------
 t
(1 row)
```

The use of operators = and <> is slightly different from other operators. If all columns in the two rows are not empty and meet the operator condition, the two

rows meet the operator condition. If any column in the two rows is not empty and does not meet the operator condition, the two rows do not meet the operator condition. If any column in the two rows is empty, the comparison result is null.

Example:

```
gaussdb=# SELECT (1,2,3) = (1,2,3) AS result;
 result
--------
 t
(1 row)

gaussdb=# SELECT (1,2,3) <> (2,2,3) AS result;
 result
--------
 t
(1 row)

gaussdb=# SELECT (2,2,3) <> (2,2,null) AS result;
 result
--------

(1 row)

gaussdb=# SELECT (null,5,6) <> (null,5,6) AS result;
 result
--------

(1 row)
```

7.7.6 Time Interval Expressions

Syntax: INTERVAL EXPR UNIT

Description: **EXPR** indicates a value, and the **UNIT** specifier indicates a unit of the value, such as HOUR, DAY, and WEEK. The keyword INTERVAL and specifier are case insensitive.

Table 7-166 describes the value range of **UNIT** in a time interval expression. Any punctuation-separated EXPR format is allowed. The recommended separators are displayed in Table 7-166.

NOTE

The INTERVAL expression supports the preceding functions only when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **5.7**, and **b_format_dev_version** is set to **s1**.

Table 7-166 Value range of UNIT in a time interval expression

UNIT Value Range	Expected EXPR Format
MICROSECOND	MICROSECONDS
SECOND	SECONDS
MINUTE	MINUTES
HOUR	HOURS
DAY	DAYS
WEEK	WEEKS

UNIT Value Range	Expected EXPR Format
MONTH	MONTHS
QUARTER	QUARTERS
YEAR	YEARS
SECOND_MICROSECOND	'SECOND_MICROSECOND'
MINUTE_MICROSECOND	'MINUTES:SECONDS.MICROSECONDS'
MINUTE_SECOND	'MINUTES:SECONDS'
HOUR_MICROSECOND	'HOURS:MINUTES:SECONDS.MICROSECONDS'
HOUR_SECOND	'HOURS:MINUTES:SECONDS'
HOUR_MINUTE	'HOURS:MINUTES'
DAY_MICROSECOND	'DAYS HOURS:MINUTES:SECONDS.MICROSECONDS'
DAY_SECOND	'DAYS HOURS:MINUTES:SECONDS'
DAY_MINUTE	'DAYS HOURS:MINUTES'
DAY_HOUR	'DAYS HOURS'
YEAR_MONTH	'YEAR_MONTH'

Example:

```
gaussdb=# SELECT DATE_ADD('2018-05-01', INTERVAL 1 DAY);
  date_add
------------
 2018-05-02
(1 row)

gaussdb=# SELECT DATE_SUB('2018-05-01', INTERVAL 1 YEAR);
  date_sub
------------
 2017-05-01
(1 row)

gaussdb=# SELECT DATE'2023-01-10' - INTERVAL 1 DAY;
      ?column?
---------------------
 2023-01-09 00:00:00
(1 row)

gaussdb=# SELECT DATE'2023-01-10' + INTERVAL 1 MONTH;
      ?column?
---------------------
 2023-02-10 00:00:00
(1 row)
```


7.8 Pseudocolumn

ROWNUM is a pseudocolumn that returns a number indicating the row number of the obtained query result. The value of **ROWNUM** in the first row is **1**, the value of **ROWNUM** in the second row is **2**, and so on. **ROWNUM** can be used to limit the number of rows returned by a query, as shown in the following example:

```
gaussdb=# CREATE TABLE Students (name varchar(20), id int) with (STORAGE_TYPE = USTORE);
gaussdb=# INSERT INTO Students VALUES ('Jack', 35);
gaussdb=# INSERT INTO Students VALUES ('Leon', 15);
gaussdb=# INSERT INTO Students VALUES ('James', 24);
gaussdb=# INSERT INTO Students VALUES ('Taker', 81);
gaussdb=# INSERT INTO Students VALUES ('Mary', 25);
gaussdb=# INSERT INTO Students VALUES ('Rose', 64);
gaussdb=# INSERT INTO Students VALUES ('Perl', 18);
gaussdb=# INSERT INTO Students VALUES ('Under', 57);
gaussdb=# INSERT INTO Students VALUES ('Angel', 101);
gaussdb=# INSERT INTO Students VALUES ('Frank', 20);
gaussdb=# INSERT INTO Students VALUES ('Charlie', 40);

-- Output the first 10 rows of data records in the Students table.
gaussdb=# SELECT * FROM Students WHERE rownum <= 10;
 name  | id
-------+-----
 Jack  |  35
 Leon  |  15
 James |  24
 Taker |  81
 Mary  |  25
 Rose  |  64
 Perl  |  18
 Under |  57
 Angel | 101
 Frank |  20
(10 rows)
```

If the statement has a clause, the output rows are reordered according to the clause.

```
gaussdb=# SELECT * FROM Students WHERE rownum < 5 order by 1;
 name  | id
-------+----
 Jack  | 35
 James | 24
 Leon  | 15
 Taker | 81
(4 rows)
```

If a subquery has a clause but the condition is placed in the outermost query, you can use the **ROWNUM** condition after sorting.

```
gaussdb=# SELECT rownum, * FROM (SELECT * FROM Students order by 1) WHERE rownum <= 2;
 rownum |  name   | id
--------+---------+-----
      1 | Angel   | 101
      2 | Charlie |  40
(2 rows)
```

As long as **ROWNUM** is greater than a specific positive integer, the condition is always false. As shown in the following example, the statement does not return any result in the table:

```
gaussdb=# SELECT * FROM Students WHERE rownum > 1;
 name | id
------+----
(0 rows)
```

Use **ROWNUM** to assign a value to each row within a certain range of the table.


```
gaussdb=# SELECT * FROM Students;
  name   | id
---------+-----
 Jack    |  35
 Leon    |  15
 James   |  24
 Taker   |  81
 Mary    |  25
 Rose    |  64
 Perl    |  18
 Under   |  57
 Angel   | 101
 Frank   |  20
 Charlie |  40
(11 rows)

gaussdb=# UPDATE Students set id = id + 5 WHERE rownum < 4;
UPDATE 3
gaussdb=# SELECT * FROM Students;
  name   | id
---------+-----
 Jack    |  40
 Leon    |  20
 James   |  29
 Taker   |  81
 Mary    |  25
 Rose    |  64
 Perl    |  18
 Under   |  57
 Angel   | 101
 Frank   |  20
 Charlie |  40
(11 rows)

gaussdb=# DROP TABLE Students;
DROP TABLE
```

The restrictions on using **ROWNUM** are as follows:

- Do not use ROWNUM as an alias to avoid ambiguity in SQL statements.
- Do not use ROWNUM when creating an index.
- Do not use ROWNUM as the default value when creating a table.
- Do not use ROWNUM as an alias in the WHERE clause.
- Do not use ROWNUM when inserting data.
- Do not use ROWNUM in a tableless query.
- Do not use ROWNUM in the LIMIT clause.
- **ROWNUM** cannot be used as a parameter of the EXECUTE statement.
- Do not use ROWNUM to update a clause in the UPSERT statement.
- If the HAVING clause contains ROWNUM (not in an aggregate function), the GROUP BY clause must also contain ROWNUM (not in an aggregate function), unless the GROUP BY clause contains an expression, for example, **SELECT a + a FROM t group by a + a having rownum < 5**.
- If the ROWNUM condition exists in the HAVING clause, the HAVING clause cannot be pushed down to any scan node.

```
gaussdb=# CREATE TABLE test (a int, b int);
CREATE TABLE
gaussdb=# INSERT INTO test SELECT generate_series, generate_series FROM generate_series(1, 10);
INSERT 0 10

-- The rownum condition cannot be pushed down to seqscan.
gaussdb=# EXPLAIN SELECT a,rownum FROM test group by a,rownum having rownum < 5;
                            QUERY PLAN
```



```
------------------------------------------------------------------------
 HashAggregate  (cost=42.23..69.10 rows=2149 width=4)
   Group By Key: a, ROWNUM
   Filter: ((ROWNUM) < 5)
   ->  Rownum  (cost=0.00..31.49 rows=2149 width=4)
         ->  Seq Scan on test  (cost=0.00..31.49 rows=2149 width=4)
(5 rows)
```

- If a subquery contains the ROWNUM condition, the predicate cannot be pushed down to any scan node.

```
-- b<5 cannot be pushed down to seqscan.
gaussdb=# EXPLAIN SELECT * FROM (SELECT * FROM test WHERE rownum < 5) WHERE b < 5;
                              QUERY PLAN
------------------------------------------------------------------------------
 Subquery Scan on __unnamed_subquery__  (cost=0.00..0.01 rows=1 width=8)
   Filter: (__unnamed_subquery__.b < 5)
   ->  Rownum  (cost=0.00..0.00 rows=1 width=8)
         StopKey: (ROWNUM < 5)
         ->  Seq Scan on test  (cost=0.00..31.49 rows=2149 width=8)
(5 rows)

gaussdb=# DROP TABLE test;
DROP TABLE
```

CAUTION

It is not recommended that the ROWNUM condition be used in the JOIN ON clause. In GaussDB, when the ROWNUM condition is used in the JOIN ON clause, the behavior in the LEFT JOIN, RIGHT JOIN, FULL JOIN, and MERGE INTO scenarios is different from that in other databases, causing risks in service migration.

If the parent query contains the **rownum** restriction and the projection column of the subquery contains **rownum**, the restriction is pushed down to the subquery. The constraints are as follows:

- The parent query can be pushed down only when the **rownum** restriction condition is <, <=, or = and the subquery directly uses **rownum** as the pseudocolumn.
- If the parent query has multiple filter criteria for **rownum** in the subquery and the pushdown requirements are met, only the first filter criterion is pushed down.
- If a subquery contains volatile functions and stored procedures, the pushdown is not supported.

7.9 Type Conversion

7.9.1 Overview

Context

SQL is a typed language. That is, every data item has an associated data type which determines its behavior and allowed usage. GaussDB has an extensible type system that is more general and flexible than other SQL implementations. Hence,

most type conversion behaviors in GaussDB are governed by general rules. This allows the use of mixed-type expressions.

The GaussDB scanner/parser divides lexical elements into five fundamental categories: integers, floating-point numbers, strings, identifiers, and keywords. Constants of most non-numeric types are first classified as strings. The SQL language definition allows specifying type names with constant strings. Example:

```
gaussdb=# SELECT text 'Origin' AS "label", point '(0,0)' AS "value";
 label  | value
--------+-------
 Origin | (0,0)
(1 row)
```

In this example, there are two literal constants of the text and point types, respectively. If a type is not specified for a string literal, then the placeholder type unknown is assigned initially.

There are four fundamental SQL constructs requiring distinct type conversion rules in GaussDB parser:

- Function calls

  Much of the SQL type system is built around a rich set of functions. Functions can have one or more arguments. Since SQL permits function overloading, the function name alone does not uniquely identify the function to be called. The parser must select the right function based on the data types of the supplied arguments.

- Operators

  SQL allows expressions with prefix and postfix unary (one-argument) operators, as well as binary (two-argument) operators. Like functions, operators can be overloaded, so the same problem of selecting the right operator exists.

- Value storage

  The INSERT and UPDATE statements place the results of expressions into a table. The expressions in the statement must be matched up with, and perhaps converted to, the types of the target columns.

- UNION, CASE, and Related Constructs

  Since all query results from a unionized SELECT statement must appear in a single set of columns, the types of the results of each SELECT clause must be matched up and converted to a uniform set. Similarly, the result expressions of a CASE construct must be converted to a common type so that the CASE expression as a whole has a known output type. The same holds for ARRAY constructs, and for the GREATEST and LEAST functions.

The system catalog pg_cast stores information about which conversions, or casts, exist between which data types, and how to perform those conversions. For details, see **PG_CAST**.

The return type and conversion behavior of an expression are determined during semantic analysis. Data types are divided into several basic type categories, including Boolean, numeric, string, bitstring, datetime, timespan, geometric, and network. Within each category there can be one or more preferred types, which are preferred when there is a choice of possible types. With careful selection of preferred types and available implicit casts, it is possible to ensure that ambiguous

expressions (those with multiple candidate parsing solutions) can be resolved in a useful way.

All type conversion rules are designed based on the following principles:

- Implicit conversions should never have surprising or unpredictable outcomes.
- There should be no extra overhead in the parser or executor if a query does not need implicit type conversion. That is, if a query is well-formed and the types already match, then the query should execute without spending extra time in the parser and without introducing unnecessary implicit conversion calls in the query.
- Additionally, if a query usually requires an implicit conversion for a function, and if then the user defines a new function with the correct argument types, the parser should use this new function.
- XML data does not support implicit type conversion, including implicit conversion between strings and XML types.

7.9.2 Operators

Operator Type Resolution

1. Select the operators to be considered from the pg_operator system catalog. Considered operators are those with the matching name and argument count. If the search path finds multiple available operators, only the most suitable one is considered.
2. Look for the best match.
   a. Discard candidate operators for which the input types do not match and cannot be converted (using an implicit conversion) to match. Unknown text can be converted to anything for this purpose. If only one candidate remains, use it; else continue to the next step.
   b. Run through all candidates and keep those with the most exact matches on input types. In this case, the domain types are considered the same as their basic types. Keep all candidates if none survives these tests. If only one candidate remains, use it; else continue to the next step.
   c. Run through all candidates and keep those that accept preferred types (of the input data type's type category) at the most positions where type conversion will be required. Keep all candidates if none accepts preferred types. If only one candidate remains, use it; else continue to the next step.
   d. If any input arguments are of unknown types, check the type categories accepted at those argument positions by the remaining candidates. At each position, select the string category if any candidate accepts that category. (This bias towards string is appropriate since an unknown-type literal looks like a string.) Otherwise, if all the remaining candidates accept the same type category, select that category; otherwise fail because the correct choice cannot be deduced without more clues. Discard candidates that do not accept the selected type category. Furthermore, if any candidate accepts a preferred type in that category, discard candidates that accept non-preferred types for that argument. Keep all candidates if none survives these tests. If only one candidate remains, use it; else continue to the next step.

e. If there are both unknown and known-type arguments, and all the known-type arguments have the same type, assume that the unknown arguments are also of that type, and check which candidates can accept that type at the unknown-argument positions. If exactly one candidate passes this test, use it. Otherwise, an error occurs.

> **CAUTION**
>
> After an operator is found, if the type of the input parameter is different from that of the operator, implicit type conversion may occur. After the conversion, unpredictable behavior may occur. If the behavior is incorrect after implicit conversion, you can use explicit type conversion to avoid this problem. For example, after the fixed-length bpchar type is converted to the variable-length text type, spaces at the end of the character string are deleted. If the character string is compared with other character strings, errors may occur.

Examples

Example 1: Use factorial operator type resolution. There is only one factorial operator (postfix !) defined in the system catalog, and it takes an argument of bigint type. The scanner assigns bigint as an initial type to the argument in this query expression:

```
gaussdb=# SELECT 40 ! AS "40 factorial";

                  40 factorial
-------------------------------------------------
 815915283247897734345611269596115894272000000000
(1 row)
```

The parser performs a type conversion on the operand and the query is equivalent to:

```
gaussdb=# SELECT CAST(40 AS bigint) ! AS "40 factorial";
```

Example 2: String concatenation operator type resolution. A string-like syntax is used for working with string types and for working with complex extension types. Strings with unspecified type are matched with likely operator candidates. An example with one unspecified argument:

```
gaussdb=# SELECT text 'abc' || 'def' AS "text and unknown";
 text and unknown
------------------
 abcdef
(1 row)
```

In this example, the parser looks for an operator whose parameters are of the text type. Such an operator is found.

Here is a concatenation of two values of unspecified types:

```
gaussdb=# SELECT 'abc' || 'def' AS "unspecified";
 unspecified
-------------
 abcdef
(1 row)
```


> **NOTE**
>
> In this case there is no initial hint for which type to use, since no types are specified in the query. So, the parser looks for all candidate operators and finds that there are candidates accepting both string-category and bit-string-category inputs. Since string category is preferred when available, that category is selected, and then the preferred type for strings, text, is used as the specific type to resolve the unknown-type literals as.

Example 3: Absolute-value and negation operator type resolution. The GaussDB operator catalog has several entries for the prefix operator @. All the entries implement absolute-value operations for various numeric data types. One of these entries is for the float8 type, which is the preferred type in the numeric category. Therefore, GaussDB will use that entry when faced with an unknown input:

```
gaussdb=# SELECT @ '-4.5' AS "abs";
 abs
-----
 4.5
(1 row)
```

Here the system has implicitly resolved the unknown-type literal as the float8 type before applying the chosen operator.

Example 4: Use the array inclusion operator type resolution as an example. Here is another example of resolving an operator with one known and one unknown input:

```
gaussdb=# SELECT array[1,2] <@ '{1,2,3}' as "is subset";
 is subset
-----------
 t
(1 row)
```

> **NOTE**
>
> The GaussDB operator catalog has several entries for the infix operator <@, but the only two that could possibly accept an integer array on the left side are array inclusion (anyarray <@ anyarray) and range inclusion (anyelement <@ anyrange). Since none of these polymorphic pseudo-types (see **Pseudo-Types**) is considered preferred, the parser cannot resolve the ambiguity on that basis. However, the last resolution rule tells it to assume that the unknown-type literal is of the same type as the other input, that is, integer array. Now only one of the two operators can match, so array inclusion is selected. (Had range inclusion been selected, we would have gotten an error, because the string does not have the right format to be a range literal.)

7.9.3 Functions

Function Type Resolution

1. Select the functions to be considered from the pg_proc system catalog. If a non-schema-qualified function name was used, the functions in the current search path are considered. If a qualified function name was given, only functions in the specified schema are considered.

   If the search path finds multiple functions of different argument types, a proper function in the path is considered.

2. Check for a function accepting exactly the input argument types. If the function exists, use it. Cases involving unknown types will never find a match at this step.

3. If no exact match is found, see if the function call appears to be a special type conversion request.
4. Look for the best match.
   a. Discard candidate functions for which the input types do not match and cannot be converted (using an implicit conversion) to match. Unknown text can be converted to anything for this purpose. If only one candidate remains, use it; else continue to the next step.
   b. Run through all candidates and keep those with the most exact matches on input types. Domains are considered the same as their base type for this purpose. Keep all candidates if none has exact matches. If only one candidate remains, use it; else continue to the next step.
   c. Run through all candidates and keep those that accept preferred types at the most positions where type conversion will be required. Keep all candidates if none accepts preferred types. If only one candidate remains, use it; else continue to the next step.
   d. If any input arguments are of unknown types, check the type categories accepted at those argument positions by the remaining candidates. At each position, select the string category if any candidate accepts that category. (This bias towards string is appropriate since an unknown-type literal looks like a string.) Otherwise, if all the remaining candidates accept the same type category, select that category; otherwise fail because the correct choice cannot be deduced without more clues. Now discard candidates that do not accept the selected type category. Furthermore, if any candidate accepts a preferred type in that category, discard candidates that accept non-preferred types for that argument. Keep all candidates if none survives these tests. If only one candidate remains, use it; else continue to the next step.
   e. If there are both unknown and known arguments, and all the known arguments have the same type, assume that the unknown arguments are also of that type, and check which candidates can accept that type at the unknown-argument positions. If exactly one candidate passes this test, use it. Otherwise, an error occurs.

Examples

Example 1: Use the rounding function argument type resolution as the first example. There is only one round function that takes two arguments; it takes a first argument of numeric type and a second argument of integer type. So the following query automatically converts the first argument of type integer to numeric:

```
gaussdb=# SELECT round(4, 4);
 round
--------
 4.0000
(1 row)
```

That query is actually transformed by the parser to:

```
gaussdb=# SELECT round(CAST (4 AS numeric), 4);
```

Since numeric constants with decimal points are initially assigned the numeric type, the following query will require no type conversion and therefore might be slightly more efficient:


```
gaussdb=# SELECT round(4.0, 4);
```

Example 2: Use the substring function type resolution as the second example. There are several substr functions, one of which takes text and integer types. If the function is called with a string constant of unspecified type, the system chooses the candidate function that accepts an argument of the preferred category string (namely of type text).

```
gaussdb=# SELECT substr('1234', 3);
 substr
--------
    34
(1 row)
```

If the string is declared to be of varchar type, as might be the case if it comes from a table, then the parser will try to convert it to become text:

```
gaussdb=# SELECT substr(varchar '1234', 3);
 substr
--------
    34
(1 row)
```

This is transformed by the parser to effectively become:

```
gaussdb=# SELECT substr(CAST (varchar '1234' AS text), 3);
```

NOTE

The parser learns from the pg_cast catalog that text and varchar are binary-compatible, meaning that one can be passed to a function that accepts the other without doing any physical conversion. Therefore, no type conversion is really inserted in this case.

And, if the function is called with an argument of integer type, the parser will try to convert that to text:

```
gaussdb=# SELECT substr(1234, 3);
substr
--------
 34
(1 row)
```

This is transformed by the parser to effectively become:

```
gaussdb=# SELECT substr(CAST (1234 AS text), 3);
 substr
--------
    34
(1 row)
```

7.9.4 Value Storage

Value Storage Type Resolution

1. Search for an exact match with the target column.
2. Try to convert the expression to the target type. This will succeed if there is a registered cast between the two types. If the expression is an unknown-type literal, the content of the literal string will be fed to the input conversion routine for the target type.
3. Check whether there is a sizing cast for the target type. A sizing cast is a cast from that type to itself. If one is found in the pg_cast catalog, apply it to the expression before storing into the destination column. The implementation

function for such a cast always takes an extra parameter of integer type. The parameter receives the destination column's **atttypmod** value (typically its declared length, although the interpretation of **atttypmod** varies for different data types), and may take a third Boolean parameter that says whether the cast is explicit or implicit. The cast function is responsible for applying any length-dependent semantics such as size checking or truncation.

Examples

Use the character storage type conversion as an example. For a target column declared as character(20), the following statement shows that the stored value is sized correctly:

```
gaussdb=# CREATE TABLE tpcds.value_storage_t1 (
    VS_COL1 CHARACTER(20)
);
gaussdb=# INSERT INTO tpcds.value_storage_t1 VALUES('abcdef');
gaussdb=# SELECT VS_COL1, octet_length(VS_COL1) FROM tpcds.value_storage_t1;
       vs_col1        | octet_length
----------------------+--------------
 abcdef               |           20
(1 row)
)

gaussdb=# DROP TABLE tpcds.value_storage_t1;
```

NOTE

What has happened here is that the two unknown literals are resolved to text by default, allowing the **||** operator to be resolved as text concatenation. Then the text result of the operator is converted to bpchar ("blank-padded char", the internal name of the character data type) to match the target column type. Since the conversion from text to bpchar is binary-coercible, this conversion does not insert any real function call. Finally, the sizing function bpchar(bpchar, integer, Boolean) is found in the system catalog and used for the operator's result and the stored column length. This type-specific function performs the required length check and addition of padding spaces.

7.9.5 UNION, CASE, and Related Constructs

SQL UNION constructs must match up possibly dissimilar types to become a single result set. The resolution algorithm is applied separately to each output column of a union query. The INTERSECT and EXCEPT constructs resolve dissimilar types in the same way as UNION. The CASE, ARRAY, VALUES, GREATEST and LEAST constructs use the identical algorithm to match up their component expressions and select a result data type.

Type Resolution for UNION, CASE, and Related Constructs

- If all inputs are of the same type and are not unknown, resolve them as the unknown type.
- If all inputs are of the unknown type, resolve them as the text type (the preferred type of the string category). Otherwise, unknown inputs are ignored.
- If the inputs are not all of the same type category, a failure will be resulted. (The unknown type is not included in this case.)
- If the inputs are all of the same type category, choose the top preferred type in that category. (Exception: The UNION operation regards the type of the first branch as the selected type.)

NOTE

typcategory in the pg_type system catalog indicates the data type category. **typispreferred** indicates whether a type is preferred in **typcategory**.

- Convert all inputs to the selected type. (Retain the original lengths of strings). Fail if there is not an implicit conversion from a given input to the selected type.
- If the input contains the json, txid_snapshot, sys_refcursor, or geometry type, UNION cannot be performed.

Type Resolution for CASE and COALESCE in TD Compatibility Type

- If all inputs are of the same type and are not unknown, resolve them as the unknown type.
- If all inputs are of the unknown type, resolve them as the text type.
- If inputs are of the string type (including unknown which is resolved as text) and digit type, resolve them as the string type. If the inputs are not of the two types, an error will be reported.
- If the inputs are all of the same type category, choose the top preferred type in that category.
- Convert all inputs to the selected type. Fail if there is not an implicit conversion from a given input to the selected type.

Type Resolution for CASE in A Compatibility Mode

decode(expr, search1, result1, search2, result2, ..., defresult): When the **sql_beta_feature** is set to **a_style_coerce**, the final return value type of the expression is set to the data type of result1 or a higher-precision data type in the same type as result1. (For example, numeric and int are both numeric data types, but numeric has higher precision and priority than int.) For CASE WHEN, the behavior is the same as the default behavior in A-compatible mode.

- If all inputs are of the same type and are not unknown, resolve them as the unknown type. Otherwise, proceed to the next step.
- Set the data type of result1 to the final return value type preferType, which belongs to preferCategory.
- Consider the data types of result2, result3, and defresult in sequence. If the type category is also preferCategory, which is the same as that of result1, check whether the precision (priority) is higher than that of preferType. If it is, update preferType to a data type with higher precision. If the type category is not preferCategory, check whether the category can be implicitly converted to preferType. If it cannot, an error is reported.
- Uses the data type recorded by preferType as the return value type of the expression. The expression result is implicitly converted to this data type.

Note 1:

There is a special case where the character type of a super-large number is converted to the numeric type, for example, **select decode(1, 2, 2, '5346546567646545465756767867 6')**, in which the large number exceeds the range of the bigint and double types. If result1 is of the numeric type and does

not meet the condition that all inputs are of the same type, the type of the return value is set to numeric to be compatible with this special case.

Note 2:

Priority of the numeric types: numeric > float8 > float4 > int8 > int4 > int2 > int1

Priority of the character types: text > varchar (nvarchar2) > bpchar > char

Priority of date types: timestamptz > timestamp > smalldatetime > date > abstime > timetz > time

Priority of date span types: interval > tinterval > reltime

Note 3:

The following figure shows the supported implicit type conversion when **set sql_beta_feature** is set to **'a_style_coerce'** in ORA-compatible mode. **** indicates that conversion is not required, **yes** indicates that conversion is supported, and the blank value indicates that conversion is not supported.

	bool	int1	int2	int4	int8	float4	float8	numeric	money	char	bpchar	varchar2	nvarchar2	text/clob	raw	blob	date	time	timetz	timestamp	timestamptz	smalldatetime	interval	reltime	abstime
bool	\																								
int1		\	yes	yes	yes	yes	yes	yes		yes	yes	yes	yes	yes											
int2		yes	\	yes	yes	yes	yes	yes		yes	yes	yes	yes	yes											
int4		yes	yes	\	yes	yes	yes	yes		yes	yes	yes	yes	yes											
int8		yes	yes	yes	\	yes	yes	yes		yes	yes	yes	yes	yes											
float4		yes	yes	yes	yes	\	yes	yes		yes	yes	yes	yes	yes											
float8		yes	yes	yes	yes	yes	\	yes		yes	yes	yes	yes	yes											
numeric		yes	yes	yes	yes	yes	yes	\		yes	yes	yes	yes	yes											
money									\																
char		yes	yes	yes	yes	yes	yes	yes		\	yes	yes	yes	yes											
bpchar		yes	yes	yes	yes	yes	yes	yes		yes	\	yes	yes	yes											
varchar2		yes	yes	yes	yes	yes	yes	yes		yes	yes	\	yes	yes	yes										
nvarchar2		yes	yes	yes	yes	yes	yes	yes		yes	yes	yes	\	yes											
text/clob		yes	yes	yes	yes	yes	yes	yes		yes	yes	yes	yes	\											
raw												yes		yes	\	yes									
blob															yes	\									
date												yes	yes	yes			\			yes	yes	yes			yes
time														yes				\	yes						
timetz														yes				yes	\						
timestamp												yes	yes	yes			yes			\	yes	yes			yes
timestamptz														yes			yes			yes	\	yes			yes
smalldatetime												yes		yes			yes			yes	yes	\			yes
interval												yes	yes	yes									\	yes	
reltime														yes									yes	\	
abstime														yes			yes			yes	yes	yes			\

Examples

Example 1: Use type resolution with underspecified types in a union as the first example. Here, the unknown-type literal **'b'** will be resolved to type **text**.

```
gaussdb=# SELECT text 'a' AS "text" UNION SELECT 'b';
 text
------
 a
 b
(2 rows)
```

Example 2: Use type resolution in a simple union as the second example. The literal **1.2** is of numeric type, and the integer value **1** can be cast implicitly to numeric type, so that type is used.

```
gaussdb=# SELECT 1.2 AS "numeric" UNION SELECT 1;
 numeric
```



```
---------
       1
     1.2
(2 rows)
```

Example 3: Use type resolution in a transposed union as the third example. Since the real type cannot be implicitly cast to integer, but integer can be implicitly cast to real, the union result type is resolved as real.

```
gaussdb=# SELECT 1 AS "real" UNION SELECT CAST('2.2' AS REAL);
 real
------
    1
  2.2
(2 rows)
```

Example 4: In TD mode, if input parameters for **COALESCE** are of int and varchar types, resolve them as the varchar type. In the A-compatible mode, an error is reported.

```
-- In the A-compatible mode, create the a_1 database compatible with A.
gaussdb=# CREATE DATABASE a_1 dbcompatibility = 'A';

-- Switch to the a_1 database.
gaussdb=# \c a_1

-- Create the t1 table.
a_1=# CREATE TABLE t1(a int, b varchar(10));

-- Show the execution plan of a statement for querying the types int and varchar of input parameters for
COALESCE.
a_1=# EXPLAIN SELECT coalesce(a, b) FROM t1;
ERROR:  COALESCE types integer and character varying cannot be matched
LINE 1: EXPLAIN SELECT coalesce(a, b) FROM t1;
                               ^
CONTEXT:  referenced column: coalesce

-- Delete the table.
a_1=# DROP TABLE t1;

-- Switch to the testdb database.
a_1=# \c testdb

-- In TD mode, create the td_1 database compatible with Teradata.
gaussdb=# CREATE DATABASE td_1 dbcompatibility = 'C';

-- Switch to the td_1 database.
gaussdb=# \c td_1

-- Create the t2 table.
td_1=# CREATE TABLE t2(a int, b varchar(10));

-- Show the execution plan of a statement for querying the types int and varchar of input parameters for
COALESCE.
td_1=# EXPLAIN VERBOSE select coalesce(a, b) from t2;
                                QUERY PLAN
-------------------------------------------------------------------------------------
 Data Node Scan  (cost=0.00..0.00 rows=0 width=0)
   Output: (COALESCE((t2.a)::character varying, t2.b))
   Node/s: All dbnodes
   Remote query: SELECT COALESCE(a::character varying, b) AS "coalesce" FROM public.t2
(4 rows)

-- Delete the table.
td_1=# DROP TABLE t2;

-- Switch to the testdb database.
td_1=# \c testdb
```



```
-- Delete the A- and TD-compatible databases.
gaussdb=# DROP DATABASE a_1;
gaussdb=# DROP DATABASE td_1;
```

Example 5: In ORA mode, set the final return value type of the expression to the data type of result1 or a higher-precision data type whose category is the same as that of the data type of result1.

```
-- In ORA mode, create the ora_1 database compatible with ORA.
gaussdb=# CREATE DATABASE ora_1 dbcompatibility = 'A';

-- Switch to the ora_1 database.
gaussdb=# \c ora_1

-- Enable the decode compatibility parameters.
set sql_beta_feature='a_style_coerce';

-- Create the t1 table.
ora_1=# CREATE TABLE t1(c_int int, c_float8 float8, c_char char(10), c_text text, c_date date);

-- Insert data.
ora_1=# INSERT INTO t1 VALUES(1, 2, '3', '4', date '12-10-2010');

-- The data type of result1 is char and that of defresult is text. The precision of text is higher, and the type of the return value is changed to text from char.
ora_1=# SELECT decode(1, 2, c_char, c_text) AS result, pg_typeof(result) FROM t1;
 result | pg_typeof
--------+-----------
 4      | text
(1 row)

-- The data type of result1 is int, which is a numeric type. The type of the return value is set to numeric.
ora_1=# SELECT decode(1, 2, c_int, c_float8) AS result, pg_typeof(result) FROM t1;
 result | pg_typeof
--------+-----------
      2 | numeric
(1 row)

-- The implicit conversion from the data type of defresult to that of result1 does not exist. If it is performed, an error is reported.
ora_1=# SELECT decode(1, 2, c_int, c_date) FROM t1;
ERROR:  CASE types integer and timestamp without time zone cannot be matched
LINE 1: SELECT decode(1, 2, c_int, c_date) FROM t1;
                                   ^
CONTEXT:  referenced column: c_date

-- Disable the decode compatibility parameters.
set sql_beta_feature='none';

-- Delete the table.
ora_1=# DROP TABLE t1;
DROP TABLE

-- Switch to the testdb database.
ora_1=# \c testdb

-- Delete the ORA-compatible database.
gaussdb=# DROP DATABASE ora_1;
DROP DATABASE
```

7.10 System Operation

GaussDB text runs SQL statements to perform different system operations, such as setting variables, displaying the execution plan, and collecting garbage data.

Setting Variables

For details about how to set various parameters for a session or transaction, see **SET**.

Displaying the Execution Plan

For details about how to display the execution plan that GaussDB makes for SQL statements, see **EXPLAIN**.

Specifying a Checkpoint in Transaction Logs

By default, WALs periodically specify checkpoints in a transaction log. **CHECKPOINT** forces an immediate checkpoint when the related command is issued, without waiting for a regular checkpoint scheduled by the system. For details, see **CHECKPOINT**.

Collecting Unnecessary Data

For details about how to collect garbage data and analyze a database as required, For details, see **VACUUM**.

Collecting Statistics

For details about how to collect statistics on tables in databases, see **ANALYZE | ANALYSE**.

Setting the Constraint Check Mode for the Current Transaction

For details about how to set the constraint check mode for the current transaction, For details, see **SET CONSTRAINTS**.

Shutting Down The Current Database Node

For details about shutting down the current database node, see **SHUTDOWN**.

7.11 Controlling Transactions

A transaction is a user-defined sequence of database operations, which form an integral unit of work.

Starting a Transaction

GaussDB starts a transaction using **START TRANSACTION** and **BEGIN**. For details, see **START TRANSACTION** and **BEGIN**.

Setting a Transaction

GaussDB sets a transaction using **SET TRANSACTION** or **SET LOCAL TRANSACTION**. For details, see **SET TRANSACTION**.

Committing a Transaction

GaussDB commits all operations of a transaction using **COMMIT** or **END**. For details, see **COMMIT | END**.

Rolling Back a Transaction

If a fault occurs during a transaction and the transaction cannot proceed, the system performs rollback to cancel all the completed database operations related to the transaction. See **ROLLBACK**.

NOTE

If an execution request (not in a transaction block) received in the database contains multiple statements, the request is packed into a transaction. If one of the statements fails, the entire request will be rolled back.

7.12 SQL Syntax

7.12.1 SQL Syntax

Table 7-167 SQL syntax

Format	Description
[ ]	The part enclosed in brackets ([]) is optional.
...	Preceding elements can appear repeatedly.
[x | y | ...]	One item is selected from two or more options or no item is selected.
{ x | y | ... }	One item is selected from two or more options.
[x | y | ...] [...]	Multiple parameters or no parameter can be selected. If multiple parameters are selected, separate them with spaces.
[x | y | ...] [,...]	Multiple parameters or no parameter can be selected. If multiple parameters are selected, separate them with commas (,).
{ x | y | ... } [...]	At least one parameter can be selected. If multiple parameters are selected, separate them with spaces.
{ x | y | ... } [,...]	At least one parameter can be selected. If multiple parameters are selected, separate them with commas (,).

7.12.2 DCL Syntax Overview

Data control language (DCL) is used to set or change the permissions of database users or roles.

Granting Rights

GaussDB provides a statement for granting rights to data objects and roles. For details, see **GRANT**.

Revoking Rights

GaussDB provides a statement for revoking rights. For details, see **REVOKE**.

Setting Default Rights

GaussDB allows users to set rights for objects that will be created. For details, see .**ALTER DEFAULT PRIVILEGES**

Changing Owners

GaussDB provides statements for changing owners of database objects. For details, see **REASSIGN OWNED**.

7.12.3 DDL Syntax Overview

Data definition language (DDL) is used to define or modify an object in a database, such as a table, an index, or a view.

NOTE

GaussDB does not support DDL operations when the CN is incomplete. For example, if the primary node of the database is faulty, creating a database or a table will fail.

Defining a Role

A role is used to manage permissions. For database security, management and operation permissions can be granted to different roles. For details about related SQL statements, see **Table 7-168**.

Table 7-168 SQL statements for defining a role

Function	SQL Statement
Creating a role	**CREATE ROLE**
Altering role attributes	**ALTER ROLE**
Dropping a role	**DROP ROLE**
Deleting the database objects owned by a database role	**DROP OWNED**

Defining a User

A user is used to log in to a database. Different permissions can be granted to users for managing data accesses and operations of the users. For details about related SQL statements, see **Table 7-169**.

Table 7-169 SQL statements for defining a user

Function	SQL Statement
Creating a User	CREATE USER
Altering user attributes	ALTER USER
Dropping a user	DROP USER

Defining a CMK

Client master keys (CMKs) are used to encrypt column encryption keys (CEKs) for the encrypted database feature. CMK definition includes creating and deleting a CMK. **Table 7-170** lists the involved SQL statements.

Table 7-170 SQL statements for defining a CMK

Function	SQL Statement
Creating a CMK	CREATE CLIENT MASTER KEY
Deleting a CMK	DROP CLIENT MASTER KEY

Defining a CEK

CEKs are used to encrypt data for the encrypted database feature. You can create a CEK, change the client master key specified by a CEK, and delete a CEK. **Table 7-170** lists the involved SQL statements.

Table 7-171 SQL statements for defining a CEK

Function	SQL Statement
Creating a CEK	CREATE COLUMN ENCRYPTION KEY
Changing the CMK specified by a CEK	ALTER COLUMN ENCRYPTION KEY
Deleting a CEK	DROP COLUMN ENCRYPTION KEY

Defining a Database

A database is the warehouse for organizing, storing, and managing data. Defining a database includes: creating a database, altering the database attributes, and dropping the database. **Table 7-172** lists the involved SQL statements.

Table 7-172 SQL statements for defining a database

Function	SQL Statement
Creating a database	CREATE DATABASE
Altering database attributes	ALTER DATABASE
Dropping a Database	DROP DATABASE

Defining a Schema

A schema is the set of a group of database objects and is used to control the access to the database objects. Table 7-173 lists the involved SQL statements.

Table 7-173 SQL statements for defining a schema

Function	SQL Statement
Creating a schema	CREATE SCHEMA
Altering schema attributes	ALTER SCHEMA
Dropping a schema	DROP SCHEMA

Defining a Tablespace

A tablespace is used to manage data objects and corresponds to a catalog on a disk. Table 7-174 lists the involved SQL statements.

Table 7-174 SQL statements for defining a tablespace

Function	SQL Statement
Creating a tablespace	CREATE TABLESPACE
Altering tablespace attributes	ALTER TABLESPACE
Dropping a tablespace	DROP TABLESPACE

Defining a Table

A table is a special data structure in a database and is used to store data objects and relationship between data objects. Table 7-175 lists the involved SQL statements.

Table 7-175 SQL statements for defining a table

Function	SQL Statement
Creating a table	**CREATE TABLE**
Altering table attributes	**ALTER TABLE**
Renaming one or more tables	**RENAME TABLE**
Dropping a table	**DROP TABLE**
Creating a table from the results of a query	**CREATE TABLE AS**

Defining a Partitioned Table

A partitioned table is a logical table used to improve query performance and does not store data (data is stored in ordinary tables). **Table 7-176** lists the involved SQL statements.

Table 7-176 SQL statements for defining a partitioned table

Function	SQL Statement
Creating a partitioned table	**CREATE TABLE PARTITION**
Create a partition	**ALTER TABLE PARTITION**
Altering partitioned table attributes	**ALTER TABLE PARTITION**
Deleting a partition	**ALTER TABLE PARTITION**
Dropping a partitioned table	**DROP TABLE**
Creating a level-2 partitioned table	**CREATE TABLE SUBPARTITION**
Modifying partitions in a level-2 partitioned table	**ALTER TABLE SUBPARTITION**

Defining an Index

An index indicates the sequence of values in one or more columns in a database table. It is a data structure that improves the speed of data access to specific information in a database table. **Table 7-177** lists the involved SQL statements.

Table 7-177 SQL statements for defining an index

Function	SQL Statement
Creating an index	**CREATE INDEX**
Altering index attributes	**ALTER INDEX**

Function	SQL Statement
Dropping an index	DROP INDEX
Rebuilding an index	REINDEX
Creating a global secondary index in a specified table	CREATE GLOBAL INDEX

Defining a Stored Procedure

A stored procedure is a set of SQL statements for achieving specific functions and is stored in the database after compiling. Users can specify a name and provide parameters (if necessary) to execute the stored procedure. Table 7-178 lists the involved SQL statements.

Table 7-178 SQL statements for defining a stored procedure

Function	SQL Statement
Creating a stored procedure	CREATE PROCEDURE
Dropping a stored procedure	DROP PROCEDURE
Recompiling a stored procedure	ALTER PROCEDURE

Defining a Function

In GaussDB, a function is similar to a stored procedure, which is a set of SQL statements. The function and stored procedure are used the same. Table 7-179 lists the involved SQL statements.

Table 7-179 SQL statements for defining a function

Function	SQL Statement
Creating a function	CREATE FUNCTION
Altering a function attribute	ALTER FUNCTION
Dropping a function	DROP FUNCTION

Defining a Package

A package consists of the package specification and package body. It is used to manage stored procedures and functions by class, which is similar to classes in languages such as Java and C++.

Table 7-180 SQL statements for defining a package

Function	SQL Statement
Creating a package	**CREATE PACKAGE**
Deleting a package	**DROP PACKAGE**
Altering a package attribute	**ALTER PACKAGE**

Defining a Cursor

To process SQL statements, the stored procedure thread assigns a memory segment to store context association. Cursors are handles or pointers to context regions. With a cursor, the stored procedure can control alterations in context areas. **Table 7-181** lists the involved SQL statements.

Table 7-181 SQL statements for defining a cursor

Function	SQL Statement
Creating a cursor	**CURSOR**
	DECLARE
Moving a cursor	**MOVE**
Closing a cursor	**CLOSE**

Defining a Resource Pool

A resource pool is a system catalog used by the resource load management module to specify attributes related to resource management, such as Cgroups. **Table 7-182** lists the involved SQL statements.

Table 7-182 SQL statements for defining a resource pool

Function	SQL Statement
Creating a resource pool	**CREATE RESOURCE POOL**
Altering resource attributes	**ALTER RESOURCE POOL**
Dropping a resource pool	**DROP RESOURCE POOL**

Defining an Aggregate Function

Table 7-183 SQL statements for defining an aggregate function

Function	SQL Statement
Creating an aggregate function	**CREATE AGGREGATE**
Modifying an aggregate function	**ALTER AGGREGATE**
Deleting an aggregate function	**DROP AGGREGATE**

Defining Data Type Conversion

Table 7-184 SQL statements for defining a data type

Function	SQL Statement
Creating user-defined data type conversion	**CREATE CAST**
Deleting user-defined data type conversion	**DROP CAST**

Defining a Plug-in Extension

This feature is for internal use only. You are advised not to use it.

Table 7-185 SQL statements for defining a plug-in extension

Function	SQL Statement
Creating an extension	**CREATE EXTENSION**
Modifying an extension	**ALTER EXTENSION**
Deleting an extension	**DROP EXTENSION**

Defining an Operator

Table 7-186 SQL statements for defining an operator

Function	SQL Statement
Creating an operator	**CREATE OPERATOR**
Modifying an operator	**ALTER OPERATOR**
Deleting an operator	**DROP OPERATOR**

Function	SQL Statement
Defining a new operator class	CREATE OPERATOR CLASS

Defining a Data Type

Table 7-187 SQL statements for defining a data type

Function	SQL Statement
Creating a data type	CREATE TYPE
Modifying a data type	ALTER TYPE
Deleting a data type	DROP TYPE

Defining a Scheduled Task

Table 7-188 SQL statements for defining a scheduled task

Function	SQL Statement
Creating a scheduled task	CREATE EVENT
Modifying a scheduled task	ALTER EVENT
Deleting a scheduled task	DROP EVENT

Defining a Database Link

DATABASE LINK can be used to remotely operate a database object. **Table 7-189** lists the involved SQL statements.

Table 7-189 Database link-related SQL statements

Function	SQL Statement
Creating a database link	CREATE DATABASE LINK
Modifying a database link	ALTER DATABASE LINK
Deleting a database link	DROP DATABASE LINK

Defining an Audit Policy

Table 7-190 SQL statements for defining an audit policy

Function	SQL Statement
Creating a unified audit policy	CREATE AUDIT POLICY
Modifying a unified audit policy	ALTER AUDIT POLICY
Deleting an audit policy	DROP AUDIT POLICY

Defining a Directory

Table 7-191 SQL statements for defining a directory

Function	SQL Statement
Creating a directory	CREATE DIRECTORY
Modifying attributes of a directory	ALTER DIRECTORY
Deleting a specified directory	DROP DIRECTORY

Defining a Foreign Data Wrapper

Table 7-192 SQL statements related to the foreign data wrapper

Function	SQL Statement
Creating a foreign data wrapper	CREATE FOREIGN DATA WRAPPER
Modifying a foreign data wrapper	ALTER FOREIGN DATA WRAPPER
Deleting a foreign data wrapper	DROP FOREIGN DATA WRAPPER

SQL Statements Related to the gs_global_config System Catalog

Table 7-193 SQL statements related to the gs_global_config system catalog

Function	SQL Statement
Adding and modifying parameter values in the gs_global_config system catalog	ALTER GLOBAL CONFIGURATION
Deleting parameter values from the gs_global_config system catalog	DROP GLOBAL CONFIGURATION

Function	SQL Statement
Inserting one or more weak passwords into the gs_global_config system catalog	CREATE WEAK PASSWORD DICTIONARY
Clearing all weak passwords in the gs_global_config system catalog	DROP WEAK PASSWORD DICTIONARY

Defining a User Group

Table 7-194 SQL statements for defining a user group

Function	SQL Statement
Creating a user group	CREATE GROUP
Modifying attributes of a user group	ALTER GROUP
Deleting a user group	DROP GROUP

Defining a Procedural Language

Table 7-195 SQL statements for defining a procedural language

Function	SQL Statement
Defining a new procedural language	CREATE LANGUAGE
Modifying the definition of a procedural language	ALTER LANGUAGE
Deleting a procedural language	DROP LANGUAGE

Defining a Masking Policy

Table 7-196 SQL statements for defining a masking policy

Function	SQL Statement
Creating a masking policy	CREATE MASKING POLICY
Modifying a masking policy	ALTER MASKING POLICY
Deleting a masking policy	DROP MASKING POLICY

Defining a Materialized View

Table 7-197 SQL statements for defining a materialized view

Function	SQL Statement
Creating a complete-refresh materialized view	CREATE MATERIALIZED VIEW
Creating a fast-refresh materialized view	CREATE INCREMENTAL MATERIALIZED VIEW
Modifying multiple auxiliary attributes of an existing materialized view	ALTER MATERIALIZED VIEW
Forcibly deleting an existing materialized view from the database	DROP MATERIALIZED VIEW
Refreshing a materialized view in complete refresh mode	REFRESH MATERIALIZED VIEW
Refreshing a materialized view in fast refresh mode	REFRESH INCREMENTAL MATERIALIZED VIEW

Defining a Resource Label

Table 7-198 SQL statements for defining a resource label

Function	SQL Statement
Creating a resource label	CREATE RESOURCE LABEL
Modifying a resource label	ALTER RESOURCE LABEL
Deleting a resource label	DROP RESOURCE LABEL

Defining a Row-Level Security Policy

Table 7-199 SQL statements for defining a row-level security policy

Function	SQL Statement
Creating a row-level security policy for a table	CREATE ROW LEVEL SECURITY POLICY
Modifying an existing row-level security policy	ALTER ROW LEVEL SECURITY POLICY
Deleting a row-level security policy from a table	DROP ROW LEVEL SECURITY POLICY

Defining a Sequence

Table 7-200 SQL statements for defining a sequence

Function	SQL Statement
Adding a sequence to the current database	CREATE SEQUENCE
Modifying parameters of an existing sequence	ALTER SEQUENCE
Deleting a sequence from the current database	DROP SEQUENCE

Defining a Foreign Server

Table 7-201 SQL statements for defining a foreign server

Function	SQL Statement
Defining a new foreign server	CREATE SERVER
Adding, modifying, and deleting parameters of an existing server	ALTER SERVER
Deleting a data server	DROP SERVER

Defining a Synonym Object

Table 7-202 SQL statements for defining a synonym object

Function	SQL Statement
Creating a synonym object	CREATE SYNONYM
Modifying the owner of a synonym object	ALTER SYNONYM
Deleting a specified synonym object	DROP SYNONYM

Defining a Trigger

Table 7-203 SQL statements for defining a trigger

Function	SQL Statement
Creating a trigger	CREATE TRIGGER

Function	SQL Statement
Renaming a trigger	ALTER TRIGGER
Deleting a trigger	DROP TRIGGER

Defining a Mapping

Table 7-204 SQL statements for defining a mapping

Function	SQL Statement
Defining a new mapping from a user to a foreign server	CREATE USER MAPPING
Altering the definition of a user mapping	ALTER USER MAPPING
Dropping a user mapping for a foreign server	DROP USER MAPPING

Defining a View

Table 7-205 SQL statements for defining a view

Function	SQL Statement
Creating a view	CREATE VIEW
Modifying the auxiliary attributes of a view	ALTER VIEW
Forcibly deleting a view from the database	DROP VIEW

Collecting Statistics

For details about how to collect statistics related to the contents of ordinary tables in the database, see **ANALYZE | ANALYSE**.

Creating an Encoding Conversion Task

For details about how to define a new conversion between two character set encodings, see **CREATE CONVERSION**.

Defining a Model

Table 7-206 SQL statements for defining a model

Function	SQL Statement
Training a machine learning model and saving the model	CREATE MODEL
Deleting a model that has been trained and saved	DROP MODEL

Defining a Rewriting Rule

Table 7-207 SQL statements for defining a rewriting rule

Function	SQL Statement
Defining a new rewriting rule	CREATE RULE
Deleting a rewriting rule	DROP RULE

Defining a Security Label

Table 7-208 SQL statements for defining a security label

Function	SQL Statement
Creating a security label	CREATE SECURITY LABEL
Applying, updating, or canceling a security label	SECURITY LABEL ON
Deleting a security label	DROP SECURITY LABEL

Importing a Database/Table

Table 7-209 SQL statements for data import

Function	SQL Statement
Preparation phase for importing a database	IMPDP DATABASE CREATE
Execution phase of importing a database	IMPDP RECOVER

Function	SQL Statement
Preparation phase for importing a table	IMPDP TABLE PREPARE
Execution phase of importing a table	IMPDP TABLE

Clearing a Recycle Bin

GaussDB provides statements for clearing a recycle bin. For details, see **PURGE**.

Clustering a Table

GaussDB supports statements for clustering a table based on an index. For details, see **CLUSTER**.

Defining an Object Comment

GaussDB supports statements for defining or modifying an object comment. For details, see **COMMENT**.

Creating a Table Based on Query Results

GaussDB supports statements for creating a table based on query results and inserting queried data into the new table. For details, see **SELECT INTO**.

Restoring a Table to an Earlier State

GaussDB allows you to restore a table to an earlier state in the event of a manual operation or application error. For details, see **TIMECAPSULE TABLE**.

Clearing Table Data

GaussDB supports statements for quickly deleting all rows from a table. For details, see **TRUNCATE**.

Recycling Storage Space

GaussDB supports statements for recycling storage space occupied by deleted rows in a table or B-Tree index. For details, see **VACUUM**.

7.12.4 DML Syntax Overview

Data manipulation language (DML) is used to perform operations on data in database tables, such as inserting, updating, querying, or deleting data.

Inserting Data

Inserting data refers to adding one or multiple records to a database table. For details, see **INSERT**.

Updating Data

Updating data refers to modifying one or multiple records in a database table. For details, see **UPDATE**.

Modifying or Inserting Data

GaussDB provides statements for matching data in a target table with that in a source table based on join conditions. If data matches, UPDATE is executed on the target table; if data does not match, INSERT is executed. For details, see **MERGE INTO**.

Querying Data

The database query statement SELECT is used to search required information in a database. For details, see **SELECT**.

Deleting Data

GaussDB provides statements for deleting data that meets specified conditions from a table. For details, see **DELETE**.

Copying Data

GaussDB provides a statement for copying data between tables and files. For details, see **COPY**.

Locks

GaussDB provides multiple lock modes to control concurrent accesses to table data. For details, see **LOCK**.

GaussDB provides bucket-level locks. For details, see **LOCK BUCKETS**.

Calling a Function

GaussDB provides three statements for calling functions. These statements are the same in the syntax structure. For details, see **CALL**.

Prepared Statements

Table 7-210 SQL statements related to prepared statements

Function	SQL Statement
Executing a prepared statement	**EXECUTE**
Deallocating a prepared statement	**DEALLOCATE**

Session Management

A session is a connection established between the user and the database. **Table 7-211** lists the related SQL statements.

Table 7-211 SQL statements related to sessions

Function	SQL Statement
Altering a session	**ALTER SESSION**
Killing a session	**ALTER SYSTEM KILL SESSION**

Executing an Anonymous Code Block

GaussDB provides statements for executing an anonymous code block. For details, see **DO**.

Exporting Files

Table 7-212 SQL statements for exporting files

Function	SQL Statement
Exporting all physical files in a database	**EXPDP DATABASE**
Exporting all files related to the table	**EXPDP TABLE**

Retrieving Data Using a Cursor

GaussDB provides statements for retrieving data using a created cursor. For details, see **FETCH**.

Inserting or Replacing Data

GaussDB provides statements for inserting or replacing data in a table. For details, see **REPLACE**.

Calculating a Value based on an Expression

GaussDB provides statements for calculating a value of a row or a group of rows based on a given value expression. For details, see **VALUES**.

Importing Data

GaussDB provides a statement for importing data from a file to a specified table in the database. For details, see **LOAD DATA**.

7.12.5 Other Syntax List

In addition to DCL, DDL, and DML syntax, GaussDB also provides syntax for other functions.

Shutting Down The Current Node

GaussDB allows users to run the **shutdown** command to shut down the current database node. For details, see **SHUTDOWN**.

SQL Statements Related to Bucket Scale-out

GaussDB supports statements used by the scale-out tool to notify the kernel of the buckets that have been migrated. For details, see **MARK BUCKETS**.

Clearing Database Connections

GaussDB supports statements for clearing database connections. For details, see **CLEAN CONNECTION**.

Showing an Execution Plan of an SQL Statement

GaussDB provides statements for showing an execution plan of an SQL statement. For details, see **EXPLAIN**.

Storage Execution Plan

GaussDB provides statements for storing query execution plan information in the **PLAN_TABLE** table. For details, see **EXPLAIN PLAN**.

Prediction:

GaussDB provides statements for using a trained model to perform inference tasks. For details, see **PREDICT BY**.

Creating a Prepared Statement

GaussDB provides statements for creating a prepared statement. For details, see **PREPARE**.

SQL Statements Related to Transactions

Table 7-213 SQL statements related to transactions

Function	SQL Statement
Rolling back the current transaction and canceling the changes in the transaction	**ABORT**
	ROLLBACK
Starting a transaction	**BEGIN**

Function	SQL Statement
	SET TRANSACTION
	START TRANSACTION
Setting transaction log checkpoints	CHECKPOINT
Committing the current transaction	COMMIT | END
Committing a prepared two-phase transaction	COMMIT PREPARED
Preparing the current transaction for two-phase commit	PREPARE TRANSACTION
Deleting a savepoint previously defined for the current transaction	RELEASE SAVEPOINT
Canceling a transaction ready for two-phase commit	ROLLBACK PREPARED
Rolling back to a savepoint	ROLLBACK TO SAVEPOINT
Creating a savepoint in the current transaction	SAVEPOINT
Setting constraint check timing for the current transaction	SET CONSTRAINTS

Modifying, Displaying, and Restoring a Runtime Parameter

Table 7-214 SQL statements for modifying, displaying, and restoring a runtime parameter

Function	SQL Statement
Modifying a runtime parameter	SET
Showing the current value of a runtime parameter	SHOW
Restoring a runtime parameter to the default value	RESET

Setting a User Identifier

Table 7-215 SQL statements for setting a user identifier

Function	SQL Statement
Setting the current user identifier of the current session	**SET ROLE**
Setting the session user identifier and the current user identifier of the current session to a specified user	**SET SESSION AUTHORIZATION**

Displaying Basic Information About All Scheduled Tasks

GaussDB provides statements for displaying basic information about all scheduled tasks in a specified schema. For details, see **SHOW EVENTS**.

Controlling Data in a Unified Manner

GaussDB provides statements for controlling data in a unified manner for multiple users. For details, see **SNAPSHOT**.

7.12.6 A

7.12.6.1 ABORT

Description

Rolls back the current transaction and cancels the changes in the transaction.

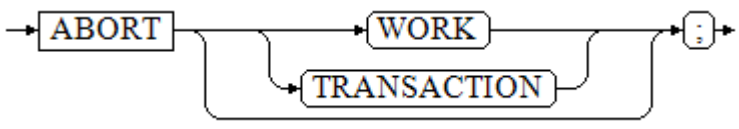
This command is equivalent to **ROLLBACK**, and is present only for historical reasons. Now ROLLBACK is recommended.

Precautions

ABORT has no impact outside a transaction, but will return a NOTICE message.

Syntax

```
ABORT [ WORK | TRANSACTION ];
```



Parameters

- **WORK | TRANSACTION**

  Optional keyword has no effect except improving readability.

Examples

```
-- Create the customer_demographics_t1 table.
gaussdb=# CREATE TABLE customer_demographics_t1
(
    CD_DEMO_SK              INTEGER              NOT NULL,
    CD_GENDER               CHAR(1)                      ,
    CD_MARITAL_STATUS       CHAR(1)                      ,
    CD_EDUCATION_STATUS     CHAR(20)                     ,
    CD_PURCHASE_ESTIMATE    INTEGER                      ,
    CD_CREDIT_RATING        CHAR(10)                     ,
    CD_DEP_COUNT            INTEGER                      ,
    CD_DEP_EMPLOYED_COUNT   INTEGER                      ,
    CD_DEP_COLLEGE_COUNT    INTEGER
)
;

-- Insert data.
gaussdb=# INSERT INTO customer_demographics_t1 VALUES(1920801,'M', 'U', 'DOCTOR DEGREE', 200,
'GOOD', 1, 0,0);

-- Start a transaction.
gaussdb=# START TRANSACTION;

-- Update the column value. Change the value of cd_education_status to Unknown.
gaussdb=# UPDATE customer_demographics_t1 SET cd_education_status= 'Unknown';

-- Abort the transaction. All updates are rolled back.
gaussdb=# ABORT;

-- Query data and find that the value of cd_education_status is not changed to Unknown.
gaussdb=# SELECT * FROM customer_demographics_t1 WHERE cd_demo_sk = 1920801;
cd_demo_sk | cd_gender | cd_marital_status | cd_education_status  | cd_purchase_estimate | cd_credit_rating
| cd_dep_count | cd_dep_employed_count | cd_dep_college_count
------------+-----------+-------------------+----------------------+----------------------+------------------
+--------------+-----------------------+----------------------
   1920801 | M         | U                 | DOCTOR DEGREE        |                  200 | GOOD             |            1
|                0 |                     0
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE customer_demographics_t1;
```

Helpful Links

SET TRANSACTION, **COMMIT | END**, and **ROLLBACK**

7.12.6.2 ALTER AGGREGATE

Description

Modifies the definition of an aggregate function, including the name, owner, and schema.

Precautions

To use ALTER AGGREGATE, you must be the owner of the aggregate function. To change the schema of an aggregate function, you must have the CREATE permission on the new schema. To change the owner, you must be a direct or indirect member of the new role, and the role must have the CREATE permission on the aggregate function's schema. (These constraints restrict the owner from performing illegal operations by deleting and rebuilding aggregate functions.)

However, a user with the SYSADMIN permission can change the ownership of an aggregate function in any way.

Syntax

```
ALTER AGGREGATE name ( argtype [ , ... ] ) RENAME TO new_name;
ALTER AGGREGATE name ( argtype [ , ... ] ) OWNER TO new_owner;
ALTER AGGREGATE name ( argtype [ , ... ] ) SET SCHEMA new_schema;
```

Parameters

- **name**

  Name (optionally schema-qualified) of an existing aggregate function.

- **argtype**

  Input data type of the aggregate function. To reference a zero-parameter aggregate function, you can write an asterisk (*) instead of a list of input data types.

- **new_name**

  New name of the aggregate function.

- **new_owner**

  New owner of the aggregate function.

- **new_schema**

  New schema of the aggregate function.

Examples

- Change the name of the aggregate function.

```
-- Create a user-defined function.
gaussdb=# CREATE OR REPLACE FUNCTION int_add(int,int)
    returns int as $BODY$
declare
begin
    return $1 + $2;
end;
$BODY$ language plpgsql;

-- Create an aggregate function.
gaussdb=# CREATE AGGREGATE myavg (int)
(
    sfunc = int_add,
    stype = int,
    initcond = '0'
);

-- Rename the aggregate function myavg that accepts int-type parameters to my_average.
gaussdb=# ALTER AGGREGATE myavg(int) RENAME TO my_average;
```

- Change the owner of the aggregate function.

```
-- Create a user joe.
gaussdb=# CREATE USER joe PASSWORD '********';

-- Change the owner of the aggregate function myavg that accepts integer-type parameters to joe.
gaussdb=# ALTER AGGREGATE my_average(integer) OWNER TO joe;
```

- Change the schema of the aggregate function.

```
-- Create a schema myschema.
gaussdb=# CREATE SCHEMA myschema;

-- Move the aggregate function myavg that accepts int-type parameters to myschema.
gaussdb=# ALTER AGGREGATE my_average(int) SET SCHEMA myschema;
```



```
-- Delete the schema, user, and related function.
gaussdb=# DROP SCHEMA myschema CASCADE;
gaussdb=# DROP USER joe;
gaussdb=# DROP FUNCTION int_add(int,int);
```

Helpful Links

CREATE AGGREGATE and **DROP AGGREGATE**

Compatibility

The SQL standard does not contain the ALTER AGGREGATE statement.

7.12.6.3 ALTER AUDIT POLICY

Description

Modifies the unified audit policy.

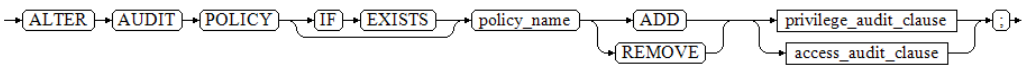
Precautions

- Only the user with the poladmin or sysadmin permission, or initial user has the permission to maintain audit policies.
- The unified audit policy takes effect only after **enable_security_policy** is set to **on**.

Syntax

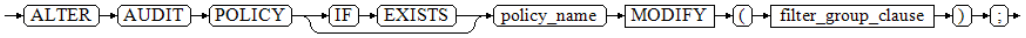
Add or delete an operation type in the audit policy.

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name { ADD | REMOVE } { privilege_audit_clause |
access_audit_clause };
```



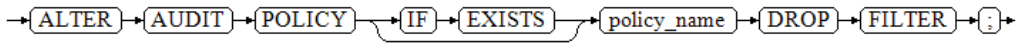
Modify the filter criteria in the audit policy.

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name MODIFY ( filter_group_clause );
```



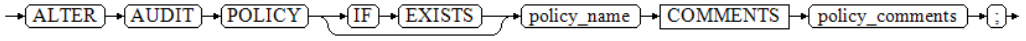
Delete the filter criteria from the audit policy.

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name DROP FILTER;
```



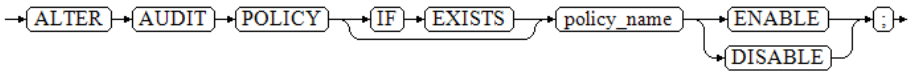
Modify the description of the audit policy.

```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name COMMENTS policy_comments;
```



Enable or disable the audit policy.

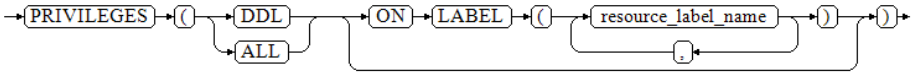
```
ALTER AUDIT POLICY [ IF EXISTS ] policy_name { ENABLE | DISABLE };
```

- privilege_audit_clause

  DDL operation type and target resource label in the audit policy.

  PRIVILEGES ({ DDL | ALL } [ON LABEL (resource_label_name [, ...])])



- access_audit_clause

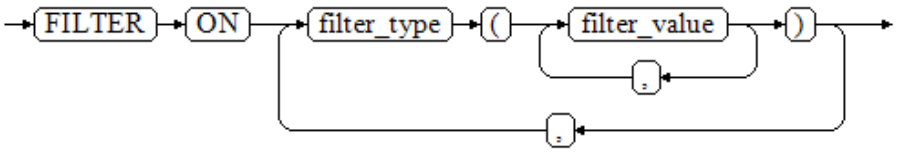
  DML operation type and target resource label in the audit policy.

  ACCESS ({ DML | ALL } [ON LABEL (resource_label_name [, ...])])

- filter_group_clause

  Filter criteria in the audit policy.

  FILTER ON { filter_type (filter_value [, ...]) } [, ...]



Parameters

- **policy_name**

  Specifies the audit policy name, which must be unique.

  Value range: a string. It must comply with the **naming convention**.

- **resource_label_name**

  Specifies the resource label name.

- **DDL**

  Specifies the operations that are audited in the database: CREATE, ALTER, DROP, ANALYZE, COMMENT, GRANT, REVOKE, SET, and SHOW.

- **DML**

  Specifies the operations that are audited in the database: SELECT, COPY, DEALLOCATE, DELETE, EXECUTE, INSERT, PREPARE, REINDEX, TRUNCATE, and UPDATE.

- **ALL**

  Specifies all operations supported by the specified DDL or DML statements in the database. When the form is { DDL | ALL }, **ALL** indicates all DDL operations. When the form is { DML | ALL }, **ALL** indicates all DML operations.

- **filter_type**

  Specifies the types of information to be filtered by the policy: **IP**, **ROLES**, and **APP**.

- **filter_value**

  Specifies the detailed information to be filtered.

- **policy_comments**

  Records description information of the audit policy.

- **ENABLE|DISABLE**

  Enables or disables the unified audit policy.

Examples

- Add or delete an operation type in the audit policy.

```
-- Create audit policy adt1 for executing CREATE on the database.
gaussdb=# CREATE AUDIT POLICY adt1 PRIVILEGES CREATE;

-- Add DROP to the adt1 audit policy.
gaussdb=# ALTER AUDIT POLICY adt1 ADD PRIVILEGES (DROP);

-- Delete DROP from the adt1 audit policy.
gaussdb=# ALTER AUDIT POLICY adt1 REMOVE PRIVILEGES (DROP);
```

- Modify the comments of the audit policy.

```
-- Change the comment of the adt1 audit policy to adt1_comments.
gaussdb=#  ALTER AUDIT POLICY adt1 COMMENTS 'adt1_comments';
```

- Modify the filter information of the audit policy.

```
-- Create a user bob_audit.
gaussdb=#  CREATE USER bob_audit PASSWORD '********';

-- Change the filtering user of the adt1 audit policy to bob_audit.
gaussdb=#  ALTER AUDIT POLICY adt1 MODIFY (FILTER ON (ROLES(bob_audit)));

-- Delete user bob_audit.
gaussdb=# DROP USER bob_audit;
```

- Delete the filter criteria from the audit policy.

```
-- Delete the filter criteria in the adt1 audit policy.
gaussdb=#  ALTER AUDIT POLICY adt1 DROP FILTER;
```

- Disable the audit policy.

```
-- Disable the adt1 audit policy.
gaussdb=#  ALTER AUDIT POLICY adt1  DISABLE;

-- Delete the adt1 audit policy.
gaussdb=# DROP AUDIT POLICY adt1;
```

Helpful Links

CREATE AUDIT POLICY and **DROP AUDIT POLICY**

7.12.6.4 ALTER COLUMN ENCRYPTION KEY

Description

Encrypts the CMKs of CEKs in round robin (RR) mode and encrypts the plaintext of CEKs.

Precautions

- This syntax is specific to the fully-encrypted database. When connecting to the database server, enable the fully-encrypted database before using this syntax.
- This syntax takes effect on CMKs only. Encrypting the plaintext of CEKs does not change the ciphertext of the encrypted columns.

Syntax

```
ALTER COLUMN ENCRYPTION KEY column_encryption_key_name WITH VALUES ( CLIENT_MASTER_KEY = client_master_key_name );
```

Parameters

- **column_encryption_key_name**

  Specifies the key name. In the same namespace, the value of this parameter must be unique.

  Value range: a string. It must comply with the **naming convention**.

- **client_master_key_name**

  Specifies the CMK used to encrypt the CEK. The value is the CMK name, which is created using the CREATE CLIENT MASTER KEY syntax. The encrypted CMKs are different from those specified before RR encryption.

NOTICE

The constraints of using Chinese cryptographic algorithms are as follows:

SM2, SM3, and SM4 are Chinese cryptographic algorithms. To avoid legal risks, these algorithms must be used together. The Chinese cryptographic algorithms used for the RR encryption must be the same as those used before RR encryption.

Helpful Links

CREATE COLUMN ENCRYPTION KEY and **DROP COLUMN ENCRYPTION KEY**

7.12.6.5 ALTER DATABASE

Description

Modifies a database, including its name, owner, object isolation, and connection limitation.

Precautions

- Only the database owner or a user granted with the ALTER permission can run the ALTER DATABASE command. By default, system administrators have the permission. The following are permission constraints depending on the attributes to be modified:
  - To modify the database name, you must have the CREATEDB permission.
  - To modify a database owner, you must be a database owner or system administrator and a member of the new owner role, with the CREATEDB permission.
  - To modify the default tablespace of the database, the user must have the CREATE permission on the tablespace. This statement physically migrates tables and indexes in a default tablespace to a new tablespace. Note that tables and indexes outside the default tablespace are not affected.

- You are not allowed to rename a database in use. To rename it, connect to another database.

Syntax

- Modify the maximum number of connections to the database.
```
ALTER DATABASE database_name
   [ WITH ] CONNECTION LIMIT connlimit;
```
- Rename the database.
```
ALTER DATABASE database_name
   RENAME TO new_name;
```
- Change the database owner.
```
ALTER DATABASE database_name
   OWNER TO new_owner;
```
- Change the default tablespace of the database.
```
ALTER DATABASE database_name
   SET TABLESPACE new_tablespace;
```

  NOTE

  If some tables or objects in the database have been created in new_tablespace, the default tablespace of the database cannot be changed to **new_tablespace**. An error will be reported during the execution.
- Modify the session parameter value of the database.
```
ALTER DATABASE database_name
   SET configuration_parameter { { TO | = } { value | DEFAULT } | FROM CURRENT };
```
- Reset the database configuration parameter.
```
ALTER DATABASE database_name RESET
   { configuration_parameter | ALL };
```
- Modify the object isolation attribute of the database.
```
ALTER DATABASE database_name [ WITH ] { ENABLE | DISABLE } PRIVATE OBJECT;
```

  NOTE

  - To modify the object isolation attribute of a database, the database must be connected. Otherwise, the modification will fail.
  - For a new database, the object isolation attribute is disabled by default. After the database object isolation attribute is enabled, the database automatically adds row-level security policies to the system catalogs PG_CLASS, PG_ATTRIBUTE, PG_PROC, PG_NAMESPACE, PGXC_SLICE, and PG_PARTITION. Common users can only view the objects (tables, functions, views, and columns) that they have the permission to access. This attribute does not take effect for administrators. After this attribute is enabled, administrators can still view all database objects.
- Change the database time zone.
```
ALTER DATABASE database_name SET DBTIMEZONE = time_zone;
```
- Migrate a specified bucket from one DN to another. The current version does not support this function.
```
ALTER DATABASE database_name
   MOVE BUCKETS(bucketlist) FROM datanode_name TO datanode_name;
```
- Enable or disable the ILM feature of the database.
```
ALTER DATABASE set ilm = { on | off } ;
```

Parameters

- **database_name**

  Specifies the name of the database whose attributes are to be modified.

  Value range: a string. It must comply with the **naming convention**.

- **connlimit**

  Specifies the maximum number of concurrent connections that can be made to this database (excluding administrators' connections).

  Value range: an integer ranging from –1 to 2^{31} – 1. You are advised to set this parameter to an integer ranging from 1 to 50. The default value **–1** indicates that there is no restriction on the number of concurrent connections.

- **new_name**

  Specifies the new name of a database.

  Value range: a string. It must comply with the **naming convention**.

- **new_owner**

  Specifies the new owner of a database.

  Value range: a string. It must be a valid username.

- **new_tablespace**

  Specifies the new default tablespace of a database. The tablespace exists in the database. The default tablespace is **pg_default**.

  Value range: a string. It must be a valid tablespace name.

- **configuration_parameter**
  - **value**

    Sets a specified database session parameter to a specified value. If the value is **DEFAULT** or **RESET**, the default setting is used in the new session. **OFF** disables the setting.

    Value range: a string.

    - **DEFAULT**
    - **OFF**
    - **RESET**
    - User-specified value: The value must meet the restriction of the modified parameter.

  - **FROM CURRENT**

    Uses the value of **configuration_parameter** of the current session.

- **time_zone**

  Sets the time zone of the database specified by **database_name**. You must have the permission on the corresponding database.

  Value range: a string.

  - Time zones supported by the system and their abbreviations.
  - -15:59~+15:00

- **RESET configuration_parameter**

  Resets the specified database session parameter.

- **RESET ALL**

  Resets all database session parameters.

NOTE

- If you modify the default tablespace of a database, the tables and indexes in the old tablespace are moved into the new tablespace. This operation does not affect the tables or indexes in other non-default tablespaces.
- The modified database session parameter values will take effect in the next session.

Examples

- Rename the database.

```
-- Create the database testdb.
gaussdb=# CREATE DATABASE testdb;

-- Rename testdb to test_db1.
gaussdb=# ALTER DATABASE testdb RENAME TO test_db1;
```

- Change the maximum number of database connections.

```
-- Change the maximum number of connections to test_db1 to 100.
gaussdb=# ALTER DATABASE test_db1 WITH CONNECTION LIMIT 100;

-- View information about test_db1.
gaussdb=# SELECT datname,datconnlimit FROM pg_database WHERE datname = 'test_db1';
 datname  | datconnlimit
----------+--------------
 test_db1 |          100
(1 row)
```

- Change the database owner.

```
-- Create user scott.
gaussdb=# CREATE USER scott PASSWORD '********';

-- Change the owner of test_db1 to scott.
gaussdb=# ALTER DATABASE test_db1 OWNER TO scott;

-- View information about test_db1.
gaussdb=# SELECT t1.datname, t2.usename
          FROM pg_database t1, pg_user t2
          WHERE t1.datname='test_db1' AND t1.datdba=t2.usesysid;
 datname  | usename
----------+---------
 test_db1 | scott
(1 row)
```

- Change the default tablespace of the database.

```
-- Create a tablespace.
gaussdb=# CREATE TABLESPACE tbs_data1 RELATIVE LOCATION 'tablespace1/tbs_data1';

-- Modify the default tablespace of test_db1.
gaussdb=# ALTER DATABASE test_db1 SET TABLESPACE tbs_data1;

-- View information about test_db1.
gaussdb=# SELECT t1.datname AS database, t2.spcname AS tablespace
          FROM pg_database t1, pg_tablespace t2
          WHERE t1.dattablespace = t2.oid AND
               t1.datname = 'test_db1';
 database | tablespace
----------+------------
 test_db1 | tbs_data1
(1 row)
```

- Modify the object isolation attribute of the database.

```
-- Create user jack.
gaussdb=# CREATE USER jack PASSWORD '********';

-- Create the test_tbl1 table in test_db1.
gaussdb=# \c test_db1
test_db1=# CREATE TABLE test_tbl1(c1 int,c2 int);

-- Switch to user jack and view pg_tables.
```



```
test_db1=# SET ROLE jack PASSWORD '********';
test_db1=> SELECT tablename FROM pg_tables WHERE tablename = 'test_tbl1';
 tablename
-----------
 test_tbl1
(1 row)

-- Modify the object isolation attribute.
test_db1=> SET ROLE scott PASSWORD '********';
test_db1=> ALTER DATABASE test_db1 ENABLE PRIVATE OBJECT;

-- Switch to user jack and view pg_tables.
test_db1=> SET ROLE jack PASSWORD '********';

-- Due to the isolation attribute, 0 data records can be queried.
test_db1=> SELECT tablename FROM pg_tables WHERE tablename = 'test_tbl1';
 tablename
-----------
(0 rows)

-- Switch to the default user and perform the deletion.
test_db1=> RESET ROLE;
test_db1=# DROP TABLE public.test_tbl1;

-- Switch to the default database. Change the database name based on actual situation.
test_db1=# \c postgres
gaussdb=# DROP DATABASE test_db1;
gaussdb=# DROP TABLESPACE tbs_data1;
gaussdb=# DROP USER jack;
gaussdb=# DROP USER scott;
```

Helpful Links

CREATE DATABASE and **DROP DATABASE**

7.12.6.6 ALTER DATABASE LINK

Description

Modifies database link objects. For details about database links, see **DATABASE LINK**.

Precautions

Currently, only the username and password can be modified for database links.

Syntax

```
ALTER [PUBLIC] DATABASE LINK dblink_name
    [CONNECT TO 'user_name' IDENTIFIED BY 'password'];
```

Parameters

- **dblink_name**

  Name of a connection.

- **user_name**

  Username for connecting to a remote database.

- **password**

  Password for connecting to a remote database.

- **PUBLIC**

  Connection type. If **PUBLIC** is not specified, the database link is private by default.

Examples

```
-- Create a user with the system administrator permission.
gaussdb=#  CREATE USER user1 WITH SYSADMIN PASSWORD '********';
gaussdb=#  SET ROLE user1 PASSWORD '********';

-- Create a public database link.
gaussdb=# CREATE PUBLIC DATABASE LINK public_dblink CONNECT TO 'user1' IDENTIFIED BY '********'
USING (host '192.168.11.11',port '54399',dbname 'db01');

-- Create a common user.
gaussdb=# CREATE USER user2 PASSWORD '********';

-- Modify database link object information.
gaussdb=# ALTER PUBLIC DATABASE LINK public_dblink CONNECT TO 'user2' IDENTIFIED BY '********';

-- Delete a public database link.
gaussdb=# DROP PUBLIC DATABASE LINK public_dblink;

-- Delete the created user.
gaussdb=# RESET ROLE;
gaussdb=# DROP USER user1;
gaussdb=# DROP USER user2;
```

Helpful Links

CREATE DATABASE LINK and **DROP DATABASE LINK**

7.12.6.7 ALTER DEFAULT PRIVILEGES

Description

ALTER DEFAULT PRIVILEGES is used to modify the default permissions of a user on a specific object in the database. It does not affect the permissions assigned to existing objects.

Precautions

Currently, you can change only the permissions for tables (including views), sequences, functions, types, CMKs of encrypted databases, and CEKs.

Syntax

```
ALTER DEFAULT PRIVILEGES
    [ FOR { ROLE | USER } target_role [, ...] ]
    [ IN SCHEMA schema_name [, ...] ]
    abbreviated_grant_or_revoke;
```

- abbreviated_grant_or_revoke grants or revokes permissions on some objects.

  ```
  grant_on_tables_clause
    | grant_on_sequences_clause
    | grant_on_functions_clause
    | grant_on_types_clause
    | grant_on_client_master_keys_clause
    | grant_on_column_encryption_keys_clause
    | revoke_on_tables_clause
    | revoke_on_sequences_clause
    | revoke_on_functions_clause
  ```



```
| revoke_on_types_clause
| revoke_on_client_master_keys_clause
| revoke_on_column_encryption_keys_clause
```

- grant_on_tables_clause grants permissions on tables.

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | ALTER | DROP |
COMMENT | INDEX | VACUUM }
   [, ...] | ALL [ PRIVILEGES ] }
   ON TABLES
   TO { [ GROUP ] role_name | PUBLIC } [, ...]
   [ WITH GRANT OPTION ]
```

- grant_on_sequences_clause grants permissions on sequences.

```
GRANT { { SELECT | UPDATE | USAGE | ALTER | DROP | COMMENT }
   [, ...] | ALL [ PRIVILEGES ] }
   ON SEQUENCES
   TO { [ GROUP ] role_name | PUBLIC } [, ...]
   [ WITH GRANT OPTION ]
```

- grant_on_functions_clause grants permissions on functions.

```
GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
   ON FUNCTIONS
   TO { [ GROUP ] role_name | PUBLIC } [, ...]
   [ WITH GRANT OPTION ]
```

- grant_on_types_clause grants permissions on types.

```
GRANT { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
   ON TYPES
   TO { [ GROUP ] role_name | PUBLIC } [, ...]
   [ WITH GRANT OPTION ]
```

- grant_on_client_master_keys_clause grants permissions on CMKs.

```
GRANT { { USAGE  | DROP } [, ...] | ALL [ PRIVILEGES ] }
   ON CLIENT_MASTER_KEYS
   TO { [ GROUP ] role_name | PUBLIC } [, ...]
   [ WITH GRANT OPTION ]
```

- grant_on_column_encryption_keys_clause grants permissions on CEKs.

```
GRANT { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
   ON COLUMN_ENCRYPTION_KEYS
   TO { [ GROUP ] role_name | PUBLIC } [, ...]
   [ WITH GRANT OPTION ]
```

- revoke_on_tables_clause revokes permissions on tables.

```
REVOKE [ GRANT OPTION FOR ]
   { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | ALTER | DROP | COMMENT |
INDEX | VACUUM }
   [, ...] | ALL [ PRIVILEGES ] }
   ON TABLES
   FROM { [ GROUP ] role_name | PUBLIC } [, ...]
   [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

- revoke_on_sequences_clause revokes permissions on sequences.

```
REVOKE [ GRANT OPTION FOR ]
   { { SELECT | UPDATE | USAGE | ALTER | DROP | COMMENT }
   [, ...] | ALL [ PRIVILEGES ] }
   ON SEQUENCES
   FROM { [ GROUP ] role_name | PUBLIC } [, ...]
   [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

- revoke_on_functions_clause revokes permissions on functions.

```
REVOKE [ GRANT OPTION FOR ]
   { {EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
   ON FUNCTIONS
   FROM { [ GROUP ] role_name | PUBLIC } [, ...]
   [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

- revoke_on_types_clause revokes permissions on types.

```
REVOKE [ GRANT OPTION FOR ]
   { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
   ON TYPES
   FROM { [ GROUP ] role_name | PUBLIC } [, ...]
   [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```


- revoke_on_client_master_keys_clause revokes permissions on CMKs.

```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON CLIENT_MASTER_KEYS
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

- revoke_on_column_encryption_keys_clause revokes permissions on CEKs.

```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON COLUMN_ENCRYPTION_KEYS
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT | CASCADE CONSTRAINTS ]
```

Parameters

- **target_role**

  Specifies the name of an existing role. If **FOR ROLE/USER** is omitted, the current role is assumed.

  Value range: an existing role name

- **schema_name**

  Specifies the name of an existing schema.

  target_role must have the CREATE permission for **schema_name**.

  Value range: an existing schema name

- **role_name**

  Specifies the name of an existing role to grant or revoke permissions for.

  Value range: an existing role name

NOTICE

To drop a role for which the default permissions have been granted, reverse the changes in its default permissions or use DROP OWNED BY to get rid of the default permission entry for the role.

Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Grant the SELECT permission on all the tables (and views) in tpcds to every user.
gaussdb=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds GRANT SELECT ON TABLES TO PUBLIC;

-- Create a common user jack.
gaussdb=# CREATE USER jack PASSWORD '********';

-- Grant the INSERT permission on all the tables in tpcds to the user jack.
gaussdb=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds GRANT INSERT ON TABLES TO jack;

-- Grant the INSERT permission on all the tables created by jack in tpcds to the user jack.
gaussdb=# GRANT USAGE,CREATE ON SCHEMA tpcds TO jack;
gaussdb=# ALTER DEFAULT PRIVILEGES FOR ROLE jack IN SCHEMA tpcds GRANT INSERT ON TABLES TO
jack;

-- Revoke the preceding permissions.
gaussdb=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds REVOKE SELECT ON TABLES FROM PUBLIC;
gaussdb=# ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds REVOKE INSERT ON TABLES FROM jack;
```



```
-- Delete user jack.
gaussdb=# DROP USER jack CASCADE;

-- Delete the schema.
gaussdb=# DROP SCHEMA tpcds;
```

Helpful Links

GRANT and **REVOKE**

7.12.6.8 ALTER DIRECTORY

Description

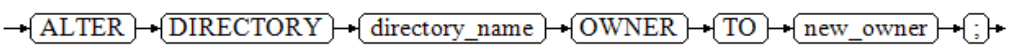
Modifies directory attributes.

Precautions

- Currently, only the directory owner can be changed.
- When **enable_access_server_directory** is set to **off**, only the initial user is allowed to change the directory owner. When **enable_access_server_directory** is set to **on**, users with the SYSADMIN permission and the directory object owner can change the directory object owner, and the user who changes the owner is required to be a member of the new owner.

Syntax

```
ALTER DIRECTORY directory_name
    OWNER TO new_owner;
```



Parameters

- **directory_name**

  Specifies the name of a directory to be modified. The value must be an existing directory name.

- **new_owner**

  Specifies the new owner of the directory.

Examples

```
-- Create a directory.
gaussdb=# CREATE OR REPLACE DIRECTORY  dir  as '/tmp/';

-- Create a user.
gaussdb=# CREATE USER jim PASSWORD '********';

-- Change the owner of the directory.
gaussdb=# ALTER DIRECTORY dir OWNER TO jim;

-- Delete a directory.
gaussdb=# DROP DIRECTORY dir;

-- Delete the user.
gaussdb=# DROP USER jim;
```


Helpful Links

CREATE DIRECTORY and **DROP DIRECTORY**

7.12.6.9 ALTER EVENT

Description

Modifies the parameters in the created scheduled event.

Precautions

- Operations related to scheduled events are supported only when **sql_compatibility** is set to **'B'**.
- Only the owner has the permission to modify the corresponding scheduled event. By default, system administrators have the permission to modify all scheduled events.
- You can execute **SHOW EVENTS** or view the **log_user** column in the PG_JOB table to obtain the event owner information.
- Each time a scheduled event is modified, the owner of the modified event is changed to the current user. If a definer is specified during modification, the owner is changed to the specified definer.
- The restrictions for the definer are the same as those described in **CREATE EVENT**.

NOTICE

If a system administrator modifies a scheduled event created by another user, the owner of the modified event is changed to the system administrator. The statements to be executed are executed by the system administrator.

Syntax

```
ALTER
    [DEFINER = user]
EVENT event_name
    [ON SCHEDULE schedule]
    [ON COMPLETION [NOT] PRESERVE]
    [RENAME TO new_event_name]
    [ENABLE | DISABLE | DISABLE ON SLAVE]
    [COMMENT 'string']
    [DO event_body]
```

- schedule

  ```
  {
      AT timestamp [+ INTERVAL interval] ...
    | EVERY interval
      [STARTS timestamp [+ INTERVAL interval] ...]
      [ENDS timestamp [+ INTERVAL interval] ...]
  }
  ```

- Interval

  ```
  quantity {YEAR | MONTH | DAY | HOUR | MINUTE | SECOND |
            YEAR_MONTH | DAY_HOUR | DAY_MINUTE |
            DAY_SECOND | HOUR_MINUTE | HOUR_SECOND |
            MINUTE_SECOND}
  ```


Parameters

- **DEFINER**

  Specifies the permission for the scheduled event statement to be executed during execution. By default, the permission of the user who creates the scheduled event is used. When definer is specified, the permission of the specified user is used.

  Only users with the sysadmin permission can specify the definer.

- **ON SCHEDULE**

  Specifies the time when a scheduled event is executed. The SCHEDULE clause is the same as that in **CREATE EVENT**.

- **RENAME TO**

  Specifies the updated scheduled event name.

- **ON COMPLETION [NOT] PRESERVE**

  Once a transaction is complete, the scheduled event is deleted from the system catalog immediately by default. You can overwrite the default behavior by setting **ON COMPLETION PRESERVE**.

- **ENABLE | DISABLE | DISABLE ON SLAVE**

  The scheduled event is in the **ENABLE** state by default after it is created. That is, the statement to be executed is executed immediately at the specified time. You can use the keyword **DISABLE** to change the **ENABLE** state. The performance of **DISABLE ON SLAVE** is the same as that of **DISABLE**.

- **COMMENT**

  You can add comments to a scheduled event. The comments can be viewed in GS_JOB_ATTRIBUTE.

- **DO**

  Specifies the statement to be executed for a scheduled event.

Examples

```
-- Create and switch to the test database.
gaussdb=# CREATE DATABASE test_event WITH DBCOMPATIBILITY = 'b';
gaussdb=# \c test_event

-- Create a table.
test_event=# CREATE TABLE t_ev(num int);

-- Create a scheduled task in the DISABLE state, indicating that the scheduled task is disabled.
test_event=# CREATE EVENT IF NOT EXISTS event_e1 ON SCHEDULE AT sysdate + interval 5 second
DISABLE DO insert into t_ev values(0);

-- To query a scheduled task, set ENABLE to f.
test_event=# SHOW EVENTS;
 job_name | schema_name | log_user | priv_user | job_status |     start_date      | interval |      end_date       |
enable | failure_msg
----------+-------------+----------+-----------+------------+---------------------+----------+---------------------
+--------+-------------
 event_e1 | public      | omm      | omm       | s          | 2023-11-28 09:10:58 | null     | 3999-12-31 16:00:00 |
f      |
(1 row)

-- Change the status of the scheduled task to ENABLE and query the table five seconds later.
test_event=# ALTER EVENT event_e1 ENABLE;
test_event=# SELECT * FROM t_ev;
 num
-----
```



```
  0
(1 row)

-- Create a scheduled event that is executed every minute.
test_event=# CREATE EVENT IF NOT EXISTS event_e2 ON SCHEDULE EVERY 1 minute DO insert into t_ev
values(1);

-- Query the table every one minute. A new record is displayed.
test_event=# SELECT * FROM t_ev;
 num
-----
  0
  1
  1
(3 rows)

-- Modify the statements to be executed in the scheduled task and query the table.
test_event=# ALTER EVENT event_e2 DO insert into t_ev values(3);

test_event=# SELECT * FROM t_ev;
 num
-----
  0
  1
  1
  3
(4 rows)

-- Change the name of a scheduled event.
test_event=# ALTER EVENT event_e2 RENAME TO event_ee;

-- Delete the scheduled event.
test_event=# DROP EVENT event_ee;

-- Drop the table.
test_event=# DROP TABLE t_ev;

-- Switch back to the initial database and delete the test database. Replace postgres with the actual
database name.
test_event=# \c postgres
gaussdb=# DROP DATABASE test_event;
```

Helpful Links

CREATE EVENT, **DROP EVENT**, and **SHOW EVENTS**

7.12.6.10 ALTER EXTENSION

NOTICE

The extended function is for internal use only. You are advised not to use it.

Description

Modifies the definition of an installed extension.

Precautions

- UPDATE

Updates the extension to a new version. The extension must be applicable to an update script (or a series of scripts) so that the current installation version can be modified to a required version.

- SET SCHEMA

  Moves the extended object to another schema. This extension must be relocatable to make the command successful.

- ADD member_object

  Adds an existing object to an extension, which is mainly applicable to the extension update script. This object is then treated as a member of the extension and can only be canceled by canceling the extension.

- DROP member_object

  Removes a member object from the extension, which is mainly applicable to the extension update script. The object is not canceled but removed from the extension.

  You must have an extension before using ALTER EXTENSION. You must have the permission on adding or deleting an object before using the ADD or DROP statement.

NOTICE

To use this function, set **support_extended_features** to **true**.

Syntax

- Modify the version of an extension.

```
ALTER EXTENSION name UPDATE [ TO new_version ];
```

- Modify the schema of the extension.

```
ALTER EXTENSION name SET SCHEMA new_schema;
```

- Add or delete member objects of the extension.

```
ALTER EXTENSION name { ADD | DROP } member_object;
```

The member object **member_object** is written as follows:

```
{AGGREGATE agg_name (agg_type [, ...] ) |
  CAST (source_type AS target_type) |
  COLLATION object_name |
  CONVERSION object_name |
  DOMAIN object_name |
  EVENT TRIGGER object_name |
  FOREIGN DATA WRAPPER object_name |

  FUNCTION function_name ( [ [ argname ] [ argmode ] argtype [, ...] ] ) |
  MATERIALIZED VIEW object_name |
  OPERATOR operator_name (left_type, right_type) |
  OPERATOR CLASS object_name USING index_method |
  OPERATOR FAMILY object_name USING index_method |
  [ PROCEDURAL ] LANGUAGE object_name |
  SCHEMA object_name |
  SEQUENCE object_name |
  SERVER object_name |
  TABLE object_name |
  TEXT SEARCH CONFIGURATION object_name |
  TEXT SEARCH DICTIONARY object_name |
  TEXT SEARCH PARSER object_name |
  TEXT SEARCH TEMPLATE object_name |
```



```
TYPE object_name |
VIEW object_name}
```

Parameters

- **name**

  Name of an installed extension.

- **new_version**

  Latest version of the extension, which can be overridden by identifiers and string literals. If a new version of the extension is not specified, ALTER EXTENSION UPDATE updates to the default version shown in the extension's control file.

- **new_schema**

  New schema of the extension.

- **object_name**

  agg_name

  function_name

  operator_name

  Names of objects that are added or removed from the extension, including names of tables, aggregations, domains, external linked lists, functions, operators, operator classes, operator families, sequences, text search objects, types, and views that can be schema-qualified.

- **agg_type**

  Input data type of the aggregate function. To reference a zero-parameter aggregate function, use ***** to replace the input data type list.

- **source_type**

  Name of the source data type to be forcibly converted.

- **target_type**

  Name of the target data type to be forcibly converted.

- **argmode**

  Model of the function parameter. The value can be **IN**, **OUT**, **INOUT**, or **VARIADIC**. The default value is **IN**. ALTER EXTENSION does not relate to the **OUT** parameter, because you only need to enter parameters to confirm the consistency of functions. Therefore, the **IN**, **INOUT**, and **VARIADIC** parameters are enough.

- **argname**

  Name of a function parameter. ALTER EXTENSION does not relate to the parameter name. Only the parameter data type is required to confirm the consistency of the function.

- **argtype**

  Data type (optionally schema-qualified) of a function parameter.

- **left_type**

  right_type

  Data type (optionally schema-qualified) of an operator parameter. **NONE** is written for a missing parameter of a prefix or suffix operator.

Examples

Update the PL/pgSQL extension to version 2.0.

```
gaussdb=# ALTER EXTENSION plpgsql UPDATE TO '2.0';
```

```
-- Create a schema.
gaussdb=# CREATE SCHEMA utils;


-- Delete a schema.
gaussdb=# DROP SCHEMA utils;
```

Update the PL/pgSQL extension mode to utils.

```
-- Create a schema.
gaussdb=# CREATE SCHEMA utils;

-- Update the PL/pgSQL extension mode to utils.
gaussdb=# ALTER EXTENSION plpgsql SET SCHEMA utils;

-- Delete a schema.
gaussdb=# DROP SCHEMA utils;
```

Add an existing function for PL/pgSQL extension.

```
gaussdb=# ALTER EXTENSION plpgsql ADD FUNCTION populate_record(anyelement, plpgsql);
```

Helpful Links

CREATE EXTENSION and **DROP EXTENSION**

7.12.6.11 ALTER FOREIGN DATA WRAPPER

Description

Modifies the definition of a foreign data wrapper.

Precautions

- Only initial users and system administrators can modify the foreign data wrapper.
- The ALTER statement can be successfully executed only when **support_extended_features** is set to **on**.

Syntax

- Set the attributes of the foreign data wrapper.
  ```
  ALTER FOREIGN DATA WRAPPER name
  [ HANDLER handler_function | NO HANDLER ]
  [ VALIDATOR validator_function | NO VALIDATOR ]
  [ OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ]) ];
  ```
- Set a new owner.
  ```
  ALTER FOREIGN DATA WRAPPER name OWNER TO new_owner;
  ```
- Set a new name.
  ```
  ALTER FOREIGN DATA WRAPPER name RENAME TO new_name;
  ```


Parameters

- **name**

  Specifies a name of an existing foreign data wrapper.

- **HANDLER handler_function**

  Specifies a new handler function for a foreign data wrapper.

- **NO HANDLER**

  Specifies that the foreign data wrapper no longer has the handler function.

  NOTICE

  Foreign tables that use foreign data wrapper but do not have handlers cannot be accessed.

- **VALIDATOR validator_function**

  Specifies a new validator function for a foreign data wrapper.

  NOTICE

  Depending on the new validator, an existing option for the foreign data wrapper or dependent server, user mapping, or foreign table may be invalid. Before using foreign data wrapper, ensure that these options are correct. However, any options specified in the **ALTER FOREIGN DATA WRAPPER** command will be checked using the new validator function.

- **NO VALIDATOR**

  Specifies that the foreign data wrapper no longer has the validator function.

- **OPTIONS ([ADD | SET | DROP] option ['value'] [, ...])**

  Modifies the options of a foreign data wrapper. ADD, SET, and DROP specify the actions to be performed. If no operation is specified, the default operation is ADD. The option name must be unique. When foreign data is used to wrap validator functions, names and values are also validated.

- **new_owner**

  Specifies the username of the new owner for a foreign data wrapper.

- **new_name**

  Specifies the new name of a foreign data wrapper.

Example

```
-- Modify a foreign data wrapper dbi, add the foo option, and delete bar.
gaussdb=# ALTER FOREIGN DATA WRAPPER dbi OPTIONS (ADD foo '1', DROP 'bar');

-- Change the validator of the foreign data wrapper dbi to bob.myvalidator.
gaussdb=# ALTER FOREIGN DATA WRAPPER dbi VALIDATOR bob.myvalidator;
```

Helpful Links

CREATE FOREIGN DATA WRAPPER and DROP FOREIGN DATA WRAPPER

7.12.6.12 ALTER FUNCTION

Description

Modifies the attributes of a user-defined function or recompiles a function.

Precautions

- Only the function owner or a user granted with the ALTER permission can run the **ALTER FUNCTION** command. By default, system administrators have the permission. The following are permission constraints depending on the attributes to be modified:
  - If a function involves operations on temporary tables, ALTER FUNCTION cannot be used.
  - To modify the owner or schema of a function, you must be a function owner or system administrator and a member of the new owner role.
  - Only the system administrator and initial user can change the schema of a function to public.
- The **plpgsql_dependency** parameter must be set for function recompilation.
- Only the initial user or the user who creates the stored procedure can modify the stored procedure to a stored procedure that has the definer permission.
- When separation of duties is enabled, no role is allowed to modify the owner of a function with the definer permission.
- When separation of duties is disabled, only the initial user and system administrator can change the owner of a function with the definer permission. However, the function owner cannot be changed to an O&M administrator.
- Only the initial user can change the owner of a function to the initial user.

Syntax

- Modify the additional parameters of the customized function.

```
ALTER FUNCTION function_name ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    action [ ... ] [ RESTRICT ];
```

  The syntax of the ACTION clause is as follows:

```
{CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT}
 | {IMMUTABLE | STABLE | VOLATILE}
 | {SHIPPABLE | NOT SHIPPABLE}
 | {NOT FENCED | FENCED}
 | [ NOT ] LEAKPROOF
 | { [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER }
 | AUTHID { DEFINER | CURRENT_USER }
 | COST execution_cost
 | ROWS result_rows
 | SET configuration_parameter { { TO | = } { value | DEFAULT }| FROM CURRENT}
 | RESET {configuration_parameter | ALL}
```

- Rename the customized function.

```
ALTER FUNCTION function_name ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    RENAME TO new_name;
```

- Change the owner of the customized function.

```
ALTER FUNCTION function_name ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    OWNER TO new_owner;
```

- Modify the schema of the customized function.


```
ALTER FUNCTION function_name ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
  SET SCHEMA new_schema;
```

- Recompile the function.

```
ALTER FUNCTION function_name COMPILE;
```

Parameters

- **function_name**

  Specifies the name of the function to be modified.

  Value range: an existing function name

- **argmode**

  Specifies whether a parameter is an input or output parameter.

  Value range:

  - **IN**: declares input parameters.
  - **OUT**: declares output parameters.
  - **INOUT**: declares input and output parameters.
  - **VARIADIC**: declares parameters of the array type.

- **argname**

  Parameter name.

  Value range: a string. It must comply with the **naming convention**.

- **argtype**

  Specifies the data type of a function parameter.

  Value range: a valid type. For details, see **Data Types**.

- **CALLED ON NULL INPUT**

  Declares that some parameters of the function can be called in normal mode if the parameter values are **NULL**. Omitting this parameter is the same as specifying it.

- **RETURNS NULL ON NULL INPUT**

  STRICT

  Specifies that the function always returns **NULL** whenever any of its parameters is **NULL**. If this parameter is specified, the function is not executed when there is **NULL** parameter; instead a **NULL** result is assumed automatically.

  RETURNS NULL ON NULL INPUT and **STRICT** have the same functions.

- **IMMUTABLE**

  Specifies that the function always returns the same result if the parameter values are the same.

- **STABLE**

  Specifies that the function cannot modify the database, and that within a single table scan it will consistently return the same result for the same parameter value, but its result varies by SQL statements.

- **VOLATILE**

  Specifies that the function value can change in a single table scan and no optimization is performed.

- **LEAKPROOF**

  Specifies that the function has no side effect and the parameter contains only the return value. **LEAKPROOF** can be set only by the system administrator.

- **EXTERNAL**

  (Optional) The purpose is to be compatible with SQL. This feature applies to all functions, not only external functions.

- **SECURITY INVOKER**

  AUTHID CURRENT_USER

  Specifies that the function will be executed with the permissions of the user who calls it. Omitting this parameter is the same as specifying it.

  SECURITY INVOKER and **AUTHID CURRENT_USER** have the same functions.

- **SECURITY DEFINER**

  AUTHID DEFINER

  Specifies that the function will be executed with the permissions of the user who created it.

  AUTHID DEFINER and **SECURITY DEFINER** have the same functions.

- **COST execution_cost**

  Estimates the execution cost of a function.

  The unit of **execution_cost** is **cpu_operator_cost**.

  Value range: a positive integer

- **ROWS result_rows**

  Estimates the number of rows returned by the function. This is only allowed when the function is declared to return a set.

  Value range: a positive number. The default value is **1000**.

- **configuration_parameter**
  - **value**

    Sets a specified database session parameter to a specified value. If the value is **DEFAULT** or **RESET**, the default setting is used in the new session. **OFF** disables the setting.

    Value range: a string

    - **DEFAULT**
    - **OFF**
    - **RESET**
    - User-specified value: The value must meet the restriction of the modified parameter.

  - **FROM CURRENT**

    Uses the value of **configuration_parameter** of the current session.

- **new_name**

  Specifies the new name of a function. To change the schema of a function, you must have the CREATE permission on the new schema.

  Value range: a string. It must comply with the **naming convention**.

- **new_owner**

  Specifies the new owner of a function. To change the owner of a function, the new owner must have the CREATE permission on the schema to which the function belongs. Note that only the initial user can set the function owner to another initial user.

  Value range: an existing user role

- **new_schema**

  Specifies the new schema of a function.

  Value range: an existing schema

Examples

- The following is an example of modifying a function:

```
-- Enable the dependency function.
gaussdb=# SET behavior_compat_options ='plpgsql_dependency';

-- Create a function.
gaussdb=# CREATE OR REPLACE FUNCTION test_func(a int) RETURN int
IS
    proc_var int;
BEGIN
    proc_var := a;
    return 1;
END;
/

-- Change the name of the function test_func(a int) to test_func_tk(a int).
gaussdb=# ALTER FUNCTION test_func(a int) RENAME TO test_func_tk;

-- Create a user jim.
gaussdb=# CREATE USER jim PASSWORD '********';

-- Change the owner of the function to jim.
gaussdb=# ALTER FUNCTION test_func_tk(a int) OWNER TO jim;

-- Create a schema named test.
gaussdb=# CREATE SCHEMA test;

-- Change the function schema to test.
gaussdb=# ALTER FUNCTION test_func_tk(a int) SET SCHEMA test;
```

- The following is an example of recompiling a function:

```
-- Recompile the function with a function name.
gaussdb=# ALTER FUNCTION test.test_func_tk COMPILE;

-- Recompile the stored procedure with a function of a signed type.
gaussdb=# ALTER FUNCTION test.test_func_tk(a int) COMPILE;

-- Delete the function.
gaussdb=# DROP FUNCTION test.test_func_tk(a int);

-- Delete user jim.
gaussdb=# DROP USER jim;

-- Delete a schema.
gaussdb=# DROP SCHEMA test;
```

Helpful Links

CREATE FUNCTION and **DROP FUNCTION**

7.12.6.13 ALTER GLOBAL CONFIGURATION

Description

Adds or modifies **key-value** of the gs_global_config system catalog. You can modify an existing parameter or add a new one.

Precautions

- Only the initial database user can run this command.
- The parameter name cannot be **weak_password** or **undostoragetype**.

Syntax

```
ALTER GLOBAL CONFIGURATION with(name=value, name=value...);
```

Parameters

- **name**

  Parameter name, which is of the text type. The value cannot be **weak_password** or **undostoragetype**.

- **value**

  Parameter value, which is of the text type.

Examples

```
-- Insert.
gaussdb=# ALTER GLOBAL CONFIGURATION with(redis_is_ok = true);

-- Query.
gaussdb=# SELECT * FROM gs_global_config;
      name       | value
-----------------+-------
 buckets_len     | 16384
 undostoragetype | page
 redis_is_ok     | true
(3 rows)

-- Modify.
gaussdb=# ALTER GLOBAL CONFIGURATION with(redis_is_ok = false);

-- Query.
gaussdb=# SELECT * FROM gs_global_config;
      name       | value
-----------------+-------
 buckets_len     | 16384
 undostoragetype | page
 redis_is_ok     | false
(3 rows)

-- Delete.
ggaussdb=# DROP GLOBAL CONFIGURATION redis_is_ok;

-- Query.
gaussdb=# SELECT * FROM gs_global_config;
      name       | value
-----------------+-------
 buckets_len     | 16384
 undostoragetype | page
(2 rows)
```


Helpful Links

DROP GLOBAL CONFIGURATION

7.12.6.14 ALTER GROUP

Description

Modifies the attributes of a user group.

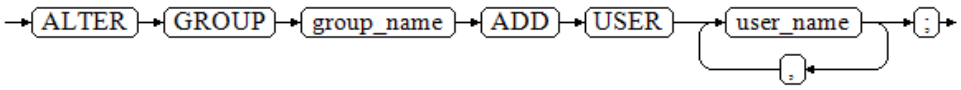
Precautions

- ALTER GROUP is not a standard SQL statement and is not recommended.
- Two clauses (ADD USER and DROP USER) are used to add users to or delete users from a user group. Any user can be a user or a user group. These two clauses are equivalent to granting or revoking the permissions of a user or role to other users or roles. Therefore, you are advised to replace them with GRANT or REVOKE.
- The RENAME TO clause changes the user group name, which is equivalent to renaming roles using ALTER ROLE.

Syntax

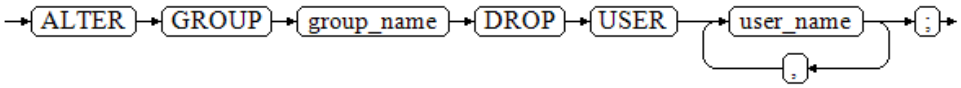
- Add users to a group.

```
ALTER GROUP group_name
    ADD USER user_name [, ... ];
```



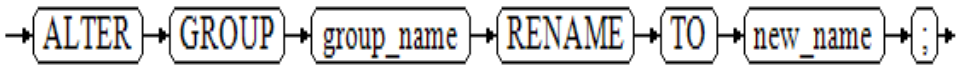
- Remove users from a group.

```
ALTER GROUP group_name
    DROP USER user_name [, ... ];
```



- Change the name of the group.

```
ALTER GROUP group_name
    RENAME TO new_name;
```



Parameters

- **user_name**

  Role name.

  Value range: an existing role name. If a role name contains uppercase letters, enclose the name with double quotation marks ("").

- **group_name**

  Name of an existing user group.

Value range: an existing role name. If a role name contains uppercase letters, enclose the name with double quotation marks ("").

- **new_name**

  Name of a new role.

  Value range: a string. It must comply with the identifier naming convention and can contain a maximum of 63 characters. If the value contains more than 63 characters, the database truncates it and retains the first 63 characters as the role name. If a role name contains uppercase letters, the database automatically converts the uppercase letters into lowercase letters. To create a role name that contains uppercase letters, enclose the role name with double quotation marks ("").

Examples

- Rename a user group.

```
-- Create a user test.
gaussdb=# CREATE ROLE test PASSWORD '********';

-- Change the username, which is equivalent to ALTER ROLE RENAME.
gaussdb=# ALTER GROUP test RENAME TO tu_a1;
```

- Add users to or delete users from a user group.

```
-- Create users tu_a2 and tu_a3.
gaussdb=# CREATE ROLE tu_a2 PASSWORD '********';
gaussdb=# CREATE ROLE tu_a3 PASSWORD '********';

-- Add user tu_a2 to user group tu_a1.
gaussdb=# ALTER GROUP tu_a1 ADD USER tu_a2;

-- The preceding SQL statement is equivalent to the GRANT statement.
gaussdb=# GRANT tu_a1 TO tu_a3;

-- Query.
gaussdb=# SELECT groname, grolist FROM pg_group WHERE groname = 'tu_a1';
 groname |    grolist
---------+---------------
 tu_a1   | {25590,25593}
(1 row)
gaussdb=# SELECT rolname, oid FROM pg_roles WHERE oid IN (25590,25593);
 rolname |  oid
---------+-------
 tu_a2   | 25590
 tu_a3   | 25593
(2 rows)

-- Delete.
gaussdb=# DROP ROLE tu_a1,tu_a2,tu_a3;
```

Helpful Links

CREATE GROUP, **DROP GROUP**, and **ALTER ROLE**

7.12.6.15 ALTER INDEX

Description

ALTER INDEX modifies the definition of an existing index.

Precautions

- Only the index owner, a user who has the INDEX permission on the table where the index resides, or a user who has the ALTER ANY INDEX permission can run the **ALTER INDEX** command. When separation of duties is disabled, system administrators have this permission by default.
- Do not keep a large number of invisible indexes on the same base table. Otherwise, the performance of DML operations such as INSERT, UPDATE, and DELETE may be affected.

Syntax

- Rename a table index.
```
ALTER INDEX [ IF EXISTS ] index_name
    RENAME TO new_name;
```
- Change the tablespace to which a table index belongs.
```
ALTER INDEX [ IF EXISTS ] index_name
    SET TABLESPACE tablespace_name;
```
- Modify the storage parameter of a table index.
```
ALTER INDEX [ IF EXISTS ] index_name
    SET ( {storage_parameter = value} [, ... ] );
```
- Reset the storage parameter of a table index.
```
ALTER INDEX [ IF EXISTS ] index_name
    RESET ( storage_parameter [, ... ] ) ;
```
- Set a table index or an index partition to be unavailable.
```
ALTER INDEX [ IF EXISTS ] index_name
    [ MODIFY PARTITION index_partition_name ] UNUSABLE;
```
- Rebuild a table index or index partition.
```
ALTER INDEX index_name
    REBUILD [ PARTITION index_partition_name ];
```
- Rename an index partition.
```
ALTER INDEX [ IF EXISTS ] index_name
    RENAME PARTITION index_partition_name TO new_index_partition_name;
```
- Modify the tablespace to which an index partition belongs.
```
ALTER INDEX [ IF EXISTS ] index_name
    MOVE PARTITION index_partition_name TABLESPACE new_tablespace;
```
- Set the distributed global secondary index to be ready.
```
ALTER INDEX [ IF EXISTS ] index_name GSIVALID;
```

  NOTE

  The centralized system does not support distributed global secondary indexes. Therefore, this syntax is not supported.

- Set the distributed global secondary index to be available.
```
ALTER INDEX [ IF EXISTS ] index_name GSIUSABLE;
```

  NOTE

  The centralized system does not support distributed global secondary indexes. Therefore, this syntax is not supported.

- Set the index to be visible.
```
ALTER INDEX [ IF EXISTS ] index_name VISIBLE;
```


NOTE

- When **disable_keyword_options** is set to **"visible"**, the VISIBLE keyword cannot be used.
- This syntax is not supported in the upgrade uncommitted phase.

- Set the index to be invisible.

```
ALTER INDEX [ IF EXISTS ] index_name INVISIBLE;
```

NOTE

- When **disable_keyword_options** is set to **"invisible"**, the INVISIBLE keyword cannot be used.
- This syntax is not supported in the upgrade uncommitted phase.
- In the standby node read scenario, after the index is set to invisible, the execution plan of the query statement may change, and the query performance of the standby node may be affected.

Parameters

- **index_name**

  Specifies the index name to be modified.

- **IF EXISTS**

  Sends a notice instead of an error if the specified index does not exist.

- **RENAME TO new_name**

  Changes only the name of the index. The stored data is not affected.

  – **new_name**

    Specifies the new name of the index.

    Value range: a string. It must comply with the **naming convention**.

- **SET TABLESPACE tablespace_name**

  Changes the index tablespace to the specified tablespace and moves index-related data files to the new tablespace.

  – **tablespace_name**

    Specifies the tablespace name.

    Value range: an existing tablespace name.

- **SET ({storage_parameter = value} [, ...])**

  Changes one or more index-method-specific storage parameters of an index. Note that the index content will not be modified immediately by this statement. You may need to use REINDEX to rebuild the index based on different parameters to achieve the expected effect.

  – **storage_parameter**

    Specifies the name of an index-method-specific parameter.
    ACTIVE_PAGES indicates the number of index pages, which may be less than the actual number of physical file pages and can be used for optimization. Currently, this parameter is valid only for the local index of the Ustore partitioned table and will be updated by VACUUM and ANALYZE (including AUTOVACUUM). You are advised not to manually set this parameter.

  – **value**

Specifies the new value for an index-method-specific storage parameter. This might be a number or a word depending on the parameter.

- **RESET ({ storage_parameter } [, ...])**

  Resets one or more index-method-specific storage parameters of an index to the default value. Similar to the SET statement, REINDEX may be used to completely update the index.

- **[MODIFY PARTITION index_partition_name] UNUSABLE**

  Sets the indexes on a table or index partition to be unavailable.

- **REBUILD [PARTITION index_partition_name]**

  Rebuilds indexes on a table or an index partition. If the index contains the **lpi_parallel_method** option and the value is **PARTITION** when the index is rebuilt, and the **parallel_workers** value of the index's table is greater than 0, the index cannot be rebuilt in parallel. If the index does not contain the **lpi_parallel_method** option or the value of the option is set to **AUTO**, page-level parallel index is rebuilt by default. For details, see **LPI_PARALLEL_METHOD**.

- **RENAME PARTITION index_partition_name TO new_index_partition_name**

  Renames an index partition.

- **MOVE PARTITION index_partition_name TABLESPACE new_tablespace**

  Modifies the tablespace to which an index partition belongs.

- **new_index_partition_name**

  Specifies the new name of the index partition.

- **index_partition_name**

  Specifies the name of an index partition.

- **new_tablespace**

  Specifies a new tablespace.

- **GSIVALID**

  Internally called by the CREATE GLOBAL INDEX CONCURRENTLY function to modify the status of distributed global secondary indexes. The centralized system does not support distributed global secondary indexes. Therefore, this syntax is not supported.

- **GSIUSABLE**

  Internally called by the VACUUM FULL function to modify the status of distributed global secondary indexes. The centralized system does not support distributed global secondary indexes. Therefore, this syntax is not supported.

- **VISIBLE**

  Sets the index to be visible.

- **INVISIBLE**

  Sets the index to be invisible.

Examples

- Rename an index.

```
-- Create the test1 table and create an index for it.
gaussdb=# CREATE TABLE test1(col1 INT, col2 INT);
gaussdb=# CREATE INDEX aa ON test1(col1);
```



```
-- Rename index aa to idx_test1_col1.
gaussdb=# ALTER INDEX aa RENAME TO idx_test1_col1;

-- Query the index information in the test1 table.
gaussdb=# SELECT tablename,indexname,tablespace FROM pg_indexes WHERE tablename = 'test1';
 tablename |    indexname    | tablespace
-----------+-----------------+------------
 test1     | idx_test1_col1 |
(1 row)
```

- Change the tablespace to which the index belongs.

```
-- Create the tbs_index1 tablespace.
gaussdb=# CREATE TABLESPACE tbs_index1 RELATIVE LOCATION 'tablespace1/tbs_index1';

-- Change the tablespace to which the idx_test1_col1 index belongs to tbs_index1.
gaussdb=# ALTER INDEX IF EXISTS idx_test1_col1 SET TABLESPACE tbs_index1;

-- Query the index information in the test1 table.
gaussdb=# SELECT tablename,indexname,tablespace FROM pg_indexes WHERE tablename = 'test1';
 tablename |    indexname    | tablespace
-----------+-----------------+------------
 test1     | idx_test1_col1 | tbs_index1
(1 row)
```

- Modify and reset index storage parameters.

```
-- View details about the idx_test1_col1 index.
gaussdb=# \di idx_test1_col1
                    List of relations
 Schema |      Name      | Type  | Owner | Table | Storage
--------+----------------+-------+-------+-------+---------
 public | idx_test1_col1 | index | omm   | test1 | {storage_type=USTORE}
(1 row)

-- Modify the fill factor of the idx_test1_col1 index.
gaussdb=# ALTER INDEX IF EXISTS idx_test1_col1 SET (FILLFACTOR = 70);

-- View details about the idx_test1_col1 index.
gaussdb=# \di idx_test1_col1
                       List of relations
 Schema |      Name      | Type  | Owner | Table |     Storage
--------+----------------+-------+-------+-------+-----------------
 public | idx_test1_col1 | index | omm   | test1 | {storage_type=USTORE,fillfactor=70}
(1 row)
-- Reset the storage parameter of the idx_test1_col1 index.
gaussdb=# ALTER INDEX IF EXISTS idx_test1_col1 RESET (FILLFACTOR);

-- View details about the idx_test1_col1 index.
gaussdb=# \di idx_test1_col1
                    List of relations
 Schema |      Name      | Type  | Owner | Table | Storage
--------+----------------+-------+-------+-------+---------
 public | idx_test1_col1 | index | omm   | test1 | {storage_type=USTORE}
(1 row)
```

- Modify the index availability.

```
-- Set the idx_test1_col1 index to be unavailable.
gaussdb=# ALTER INDEX IF EXISTS idx_test1_col1 UNUSABLE;

-- Check the availability of the idx_test1_col1 index.
gaussdb=# SELECT indisusable FROM pg_index WHERE indexrelid = 'idx_test1_col1'::regclass;
 indisusable
-------------
 f
(1 row)

-- Rebuild the idx_test1_col1 index.
gaussdb=# ALTER INDEX idx_test1_col1 REBUILD;

-- Check the availability of the idx_test1_col1 index.
gaussdb=# SELECT indisusable FROM pg_index WHERE indexrelid = 'idx_test1_col1'::regclass;
 indisusable
```



```
-------------
 t
(1 row)

-- Delete.
gaussdb=# DROP INDEX idx_test1_col1;
gaussdb=# DROP TABLE test1;
gaussdb=# DROP TABLESPACE tbs_index1;
```

- Rename an index partition.

```
-- Create the partitioned table test2.
gaussdb=# CREATE TABLE test2(col1 int, col2 int) PARTITION BY RANGE (col1)(
    PARTITION p1 VALUES LESS THAN (100),
    PARTITION p2 VALUES LESS THAN (200)
);

-- Create a partitioned index.
gaussdb=# CREATE INDEX idx_test2_col1 ON test2(col1) LOCAL(
    PARTITION p1,
    PARTITION p2
);
-- Rename the index partition.
gaussdb=# ALTER INDEX idx_test2_col1 RENAME PARTITION p1 TO p1_test2_idx;
gaussdb=# ALTER INDEX idx_test2_col1 RENAME PARTITION p2 TO p2_test2_idx;

-- Query the partition name of the idx_test2_col1 index.
gaussdb=# SELECT relname FROM pg_partition WHERE parentid = 'idx_test2_col1'::regclass;
   relname
--------------
 p1_test2_idx
 p2_test2_idx
(2 rows)
```

- Modify the tablespace to which an index partition belongs.

```
-- Create tablespaces tbs_index2 and tbs_index3.
gaussdb=# CREATE TABLESPACE tbs_index2 RELATIVE LOCATION 'tablespace1/tbs_index2';
gaussdb=# CREATE TABLESPACE tbs_index3 RELATIVE LOCATION 'tablespace1/tbs_index3';

-- Change the tablespace to which the idx_test2_col1 index partition belongs.
gaussdb=# ALTER INDEX idx_test2_col1 MOVE PARTITION p1_test2_idx TABLESPACE tbs_index2;
gaussdb=# ALTER INDEX idx_test2_col1 MOVE PARTITION p2_test2_idx TABLESPACE tbs_index3;

-- Query the tablespace to which the idx_test2_col1 index partition belongs.
gaussdb=# SELECT t1.relname index_name,
       t2.spcname tablespace_name
FROM pg_partition t1, pg_tablespace t2
WHERE t1.parentid = 'idx_test2_col1'::regclass AND
      t1.reltablespace = t2.oid;
  index_name  | tablespace_name
--------------+-----------------
 p1_test2_idx | tbs_index2
 p2_test2_idx | tbs_index3
(2 rows)

-- Delete.
gaussdb=# DROP INDEX idx_test2_col1;
gaussdb=# DROP TABLE test2;
gaussdb=# DROP TABLESPACE tbs_index2;
gaussdb=# DROP TABLESPACE tbs_index3;
```

Helpful Links

CREATE INDEX, **DROP INDEX**, and **REINDEX**

7.12.6.16 ALTER LANGUAGE

This version does not support this syntax.

7.12.6.17 ALTER MASKING POLICY

Description

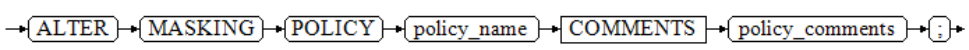
Modifies a masking policy.

Precautions

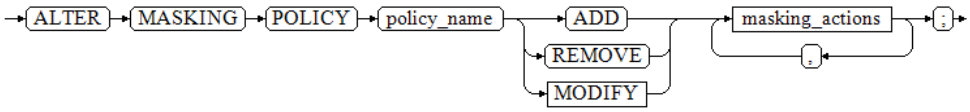
- Only users with the poladmin or sysadmin permission, or the initial user can perform this operation.
- The masking policy takes effect only after the security policy is enabled, that is, **enable_security_policy** is set to **on**.

Syntax

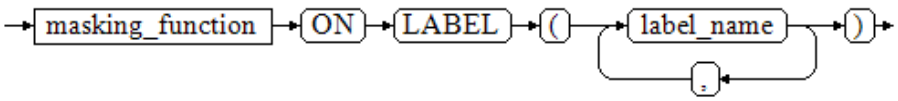
- Modify the policy description.
  ALTER MASKING POLICY policy_name COMMENTS policy_comments;

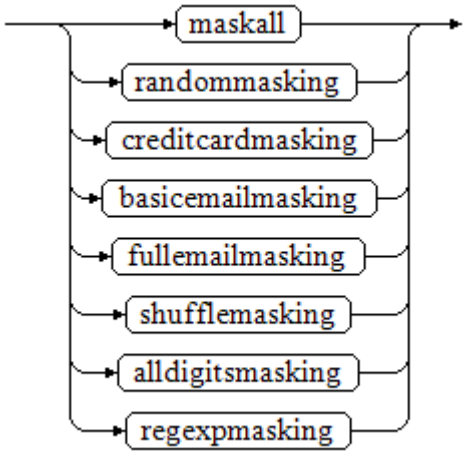
- Modify the masking method.
  ALTER MASKING POLICY policy_name {ADD | REMOVE | MODIFY} masking_actions[, ...];

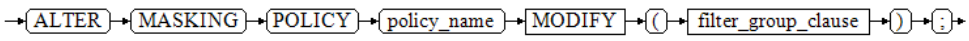
  - The syntax of **masking_actions**
    masking_function ON LABEL(label_name[, ...])

  - The syntax of **masking_function**
    {maskall | randommasking | creditcardmasking | basicemailmasking | fullemailmasking | shufflemasking | alldigitsmasking | regepmasking}

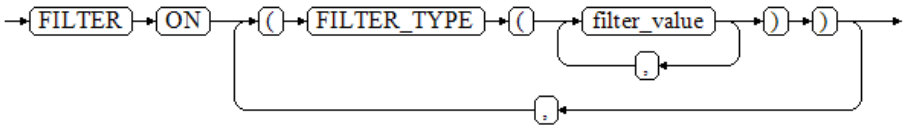
    

- Modify the scenarios where the masking policies take effect.
  ALTER MASKING POLICY policy_name MODIFY (filter_group_clause);

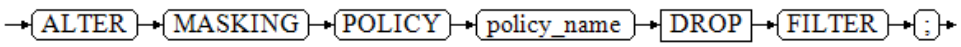
  - The syntax of **filter_group_clause**

    ```
    FILTER ON { ( FILTER_TYPE ( filter_value [, ... ] ) ) [, ... ] }
    ```

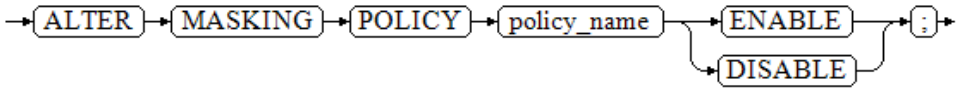
- Remove the filters of the masking policies.

  ```
  ALTER MASKING POLICY policy_name DROP FILTER;
  ```

- Enable or disable the masking policies.

  ```
  ALTER MASKING POLICY policy_name {ENABLE | DISABLE};
  ```

Parameters

- **policy_name**

  Specifies the masking policy name, which must be unique.

  Value range: a string. It must comply with the **naming convention**.

- **policy_comments**

  Adds or modifies description of masking policies.

- **masking_function**

  Specifies eight preset masking methods or user-defined functions. Schema is supported.

  maskall is not a preset function and cannot be displayed by running **\df**.

  The masking methods during presetting are as follows:

  ```
  { maskall | randommasking | creditcardmasking | basicemailmasking | fullemailmasking |
  shufflemasking | alldigitsmasking | regexpmasking }
  ```

- **label_name**

  Specifies the resource label name.

- **FILTER_TYPE**

  Specifies the types of information to be filtered by the policies: **IP**, **ROLES**, and **APP**.

- **filter_value**

  Indicates the detailed information to be filtered, such as the IP address, app name, and username.

- **ENABLE|DISABLE**

  Enables or disables the masking policy. If **ENABLE|DISABLE** is not specified, **ENABLE** is used by default.

Examples

- Modify the policy description.

  ```
  -- Create table tb_for_masking.
  gaussdb=# CREATE TABLE tb_for_masking(idx int, col1 text, col2 text, col3 text, col4 text, col5 text,
  col6 text, col7 text,col8 text);
  ```



```
-- Insert data into the tb_for_masking table.
gaussdb=# INSERT INTO tb_for_masking VALUES(1, '9876543210', 'usr321usr', 'abc@huawei.com',
'abc@huawei.com', '1234-4567-7890-0123', 'abcdef 123456 ui 323 jsfd321 j3k2l3',
'4880-9898-4545-2525', 'this is a llt case');

-- View data.
gaussdb=# SELECT * FROM tb_for_masking;
 idx |    col1    |   col2    |      col3      |      col4      |       col5         |               col6
     |        col7         |        col8
-----+------------+-----------+----------------+----------------+--------------------
+-------------------------------
------+---------------------+--------------------
   1 | 9876543210 | usr321usr | abc@huawei.com | abc@huawei.com | 1234-4567-7890-0123 | abcdef
123456 ui 323 jsfd321 j
3k2l3 | 4880-9898-4545-2525 | this is a llt case
(1 row)

-- Create a resource label for sensitive column col1.
gaussdb=# CREATE RESOURCE LABEL mask_lb1 ADD COLUMN(tb_for_masking.col1);

-- Create a data masking policy named maskpol1.
gaussdb=# CREATE MASKING POLICY maskpol1 maskall ON LABEL(mask_lb1);

-- Add description for masking policy maskpol1.
gaussdb=# ALTER MASKING POLICY maskpol1 COMMENTS 'masking policy for tb_for_masking.col1';

-- View the description of data masking policy maskpol1.
gaussdb=# SELECT * FROM GS_MASKING_POLICY;
 polname  |              polcomments               |         modifydate         | polenabled
----------+----------------------------------------+----------------------------+------------
 maskpol1 | masking policy for tb_for_masking.col1 | 2023-11-07 16:38:31.607374 | t
(1 row)
```

- Modify a masking policy.

```
-- Create a resource label for sensitive column col2.
gaussdb=# CREATE RESOURCE LABEL mask_lb2 ADD COLUMN(tb_for_masking.col2);

-- Add the randommasking masking mode to the masking policy maskpol1.
gaussdb=# ALTER MASKING POLICY maskpol1 ADD randommasking ON LABEL(mask_lb2);

-- Access the tb_for_masking table. The masking policy is triggered for the col2 column.
gaussdb=# SELECT col2 FROM tb_for_masking;
    col2
------------
 27e8da66cc
(1 row)

-- Remove the randommasking masking mode from the masking policy maskpol1.
gaussdb=# ALTER MASKING POLICY maskpol1 REMOVE randommasking ON LABEL(mask_lb2);

-- Access the tb_for_masking table. The data in the col2 column is not masked, indicating that the
masking mode randommasking is invalid.
gaussdb=# SELECT col2 FROM tb_for_masking;
   col2
-----------
 usr321usr
(1 row)

-- Change the masking mode of the masking policy maskpol1 to randommasking.
gaussdb=# ALTER MASKING POLICY maskpol1 MODIFY randommasking ON LABEL(mask_lb1);

-- Access the tb_for_masking table. The masking policy is triggered for the col1 column.
gaussdb=# SELECT col1 FROM tb_for_masking;
    col1
------------
 5a03debac1
(1 row)
```

- Modify the scenarios where the masking policies take effect.


```
-- Create users dev_mask and bob_mask.
gaussdb=# CREATE USER dev_mask PASSWORD '*********';
gaussdb=# CREATE USER bob_mask PASSWORD '*********';

-- Create a resource label for sensitive column col8.
gaussdb=# CREATE RESOURCE LABEL mask_lb8 ADD COLUMN(tb_for_masking.col8);

-- Create a data masking policy named maskpol8.
gaussdb=# CREATE MASKING POLICY maskpol8 randommasking ON LABEL(mask_lb8) FILTER ON
ROLES(dev_mask, bob_mask), APP(gsql), IP('172.31.17.160', '127.0.0.0/24');

-- Modify the filtering information by specifying ROLES of data masking policy maskpol8.
gaussdb=# ALTER MASKING POLICY maskpol1 MODIFY (FILTER ON ROLES(dev_mask));

-- Check tb_for_masking as the dev_mask user.
gaussdb=# GRANT ALL PRIVILEGES TO dev_mask;

-- Access the tb_for_masking table. The masking policy is triggered for the col8 column.
gaussdb=# SELECT col8 FROM tb_for_masking;
       col8
--------------------
 f134e06ef528013b46
(1 row)
```

- Remove the filters of the masking policies for the policies to take effect.

```
gaussdb=# ALTER MASKING POLICY maskpol1 DROP FILTER;
```

- Disable a masking policy.

```
-- Disable masking policy maskpol1.
gaussdb=# ALTER MASKING POLICY maskpol1 DISABLE;

-- Check the status of data masking policy maskpol1. If the value of polenabled is f, the data
masking policy is disabled successfully.
gaussdb=# SELECT * FROM GS_MASKING_POLICY;
 polname  | polcomments |         modifydate         | polenabled
----------+-------------+----------------------------+------------
 maskpol1 |             | 2023-11-07 17:22:54.594111 | f
```

- Delete data.

```
-- Delete a masking policy.
gaussdb=# DROP MASKING POLICY maskpol1, maskpol8;

-- Delete a resource label.
gaussdb=# DROP RESOURCE LABEL mask_lb1, mask_lb2, mask_lb8;

-- Delete the tb_for_masking table.
gaussdb=# DROP TABLE tb_for_masking;

-- Delete the dev_mask and bob_mask users.
gaussdb=# DROP USER dev_mask, bob_mask;
```

Helpful Links

CREATE MASKING POLICY and **DROP MASKING POLICY**

7.12.6.18 ALTER MATERIALIZED VIEW

Description

Modifies multiple auxiliary attributes of an existing materialized view.

Statements and actions that can be used for ALTER MATERIALIZED VIEW are a subset of ALTER TABLE and have the same meaning when used for materialized views. For details, see **ALTER TABLE**.

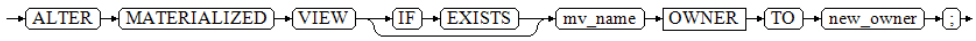
Precautions

- Only the owner of a materialized view or a system administrator has the ALTER MATERIALIZED VIEW permission.
- The materialized view structure cannot be modified.

Syntax

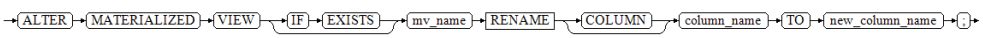
- Change the owner of a materialized view.

```
ALTER MATERIALIZED VIEW [ IF EXISTS ] mv_name
    OWNER TO new_owner;
```



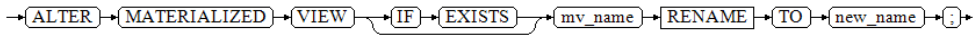
- Rename a column of a materialized view.

```
ALTER MATERIALIZED VIEW [ IF EXISTS ] mv_name
    RENAME [ COLUMN ] column_name TO new_column_name;
```



- Rename a materialized view.

```
ALTER MATERIALIZED VIEW [ IF EXISTS ] mv_name
    RENAME TO new_name;
```



Parameters

- **mv_name**

  Specifies the name of an existing materialized view, which can be schema-qualified.

  Value range: a string. It must comply with the **naming convention**.

- **column_name**

  Specifies the name of a new or existing column.

  Value range: a string. It must comply with the **naming convention**.

- **new_column_name**

  Specifies the new name of an existing column.

- **new_owner**

  Specifies the username of the new owner of a materialized view.

- **new_name**

  Specifies the new name of a materialized view.

Examples

- Change the owner of a materialized view.

```
-- Create a table.
gaussdb=# CREATE TABLE my_table (c1 int, c2 int) WITH(STORAGE_TYPE=ASTORE);

-- Create a complete-refresh materialized view.
gaussdb=# CREATE MATERIALIZED VIEW foo AS SELECT * FROM my_table;

-- Create a user.
gaussdb=# CREATE USER test PASSWORD '********';

-- Change the owner of the complete-refresh materialized view.
gaussdb=# ALTER MATERIALIZED VIEW foo OWNER TO test;
```



```
-- View the information about the materialized view.
gaussdb=# \dm foo
                        List of relations
 Schema | Name |       Type        | Owner |             Storage
--------+------+-------------------+-------+---------------------------------
 public | foo  | materialized view | test  | {orientation=row,compression=no}
(1 row)
```

- Rename a column of a materialized view.

```
-- Query the columns of the materialized view.
gaussdb=# \d foo;
Materialized view "public.foo"
 Column |  Type   | Modifiers
--------+---------+-----------
 c1     | integer |
 c2     | integer |
Rules:
    "_RETURN" AS
    ON SELECT TO foo DO INSTEAD  SELECT my_table.c1, my_table.c2
   FROM my_table
Replica Identity: NOTHING

-- Change column c1 of materialized view foo to col1 and column c2 to col2.
gaussdb=# ALTER MATERIALIZED VIEW foo RENAME c1 to col1;
gaussdb=# ALTER MATERIALIZED VIEW foo RENAME c2 to col2;

-- Use the SELECT statement to view the columns of the materialized view.
gaussdb=# SELECT * FROM foo WHERE 1=2;
 col1 | col2
------+------
(0 rows)
```

- Rename a materialized view.

```
-- Rename the materialized view foo to my_mview.
gaussdb=# ALTER MATERIALIZED VIEW foo RENAME TO my_mview;

-- Query information.
gaussdb=# \dm my_mview
                           List of relations
 Schema |   Name   |       Type        | Owner |             Storage
--------+----------+-------------------+-------+---------------------------------
 public | my_mview | materialized view | test  | {orientation=row,compression=no}
(1 row)

-- Delete.
gaussdb=# DROP MATERIALIZED VIEW my_mview;
gaussdb=# DROP TABLE my_table ;
gaussdb=# DROP USER test;
```

Helpful Links

CREATE MATERIALIZED VIEW, **CREATE INCREMENTAL MATERIALIZED VIEW**, **DROP MATERIALIZED VIEW**, **REFRESH INCREMENTAL MATERIALIZED VIEW**, and **REFRESH MATERIALIZED VIEW**

7.12.6.19 ALTER OPERATOR

Description

Modifies the definition of an operator.

Precautions

To use ALTER OPERATOR, you must be the owner of the operator. To modify the owner, you must also be a direct or indirect member of the new owning role, and

that member must have CREATE permission on the operator's schema. (These restrict that the owner cannot be changed by doing anything other than deleting or rebuilding the operator. However, a user with the SYSADMIN permission can modify the ownership of any operator in any way.)

Syntax

- Change the owner of the operator.
  ```
  ALTER OPERATOR name ( { left_type | NONE } , { right_type | NONE } ) OWNER TO new_owner;
  ```
- Change the schema of the operator.
  ```
  ALTER OPERATOR name ( { left_type | NONE } , { right_type | NONE } ) SET SCHEMA new_schema;
  ```

Parameters

- **name**

  Name of an existing operator.

- **left_type**

  Data type of the left operand for the operator; if there is no left operand, write **NONE**.

- **right_type**

  Data type of the right operand for the operator; if there is no right operand, write **NONE**.

- **new_owner**

  New owner of the operator.

- **new_schema**

  New schema name of the operator.

Example

```
-- Create a function.
gaussdb=# CREATE FUNCTION func_add(num1 integer, num2 integer) RETURN INTEGER AS
BEGIN
  RETURN num1+num2;
END;
/

-- Create an operator.
gaussdb=# CREATE OPERATOR @@@ (PROCEDURE = func_add,LEFTARG = int,RIGHTARG = int);

-- Create a user.
gaussdb=# CREATE USER user1 PASSWORD '********';

-- Modify the operator.
gaussdb=# ALTER OPERATOR @@@ (int,int) OWNER to user1;

-- Create a schema.
gaussdb=# CREATE SCHEMA oper_sch;

-- Modify the operator.
gaussdb=# ALTER OPERATOR @@@ (int,int) SET SCHEMA oper_sch;

-- Delete.
gaussdb=# DROP OPERATOR oper_sch.@@@(int,int);
gaussdb=# DROP SCHEMA oper_sch;
gaussdb=# DROP USER user1;
```


Helpful Links

CREATE OPERATOR, **CREATE OPERATOR CLASS**, and **DROP OPERATOR**

Compatibility

The SQL standard does not contain the ALTER OPERATOR statement.

7.12.6.20 ALTER PACKAGE

Description

Modifies the attributes of a package or recompiles a package.

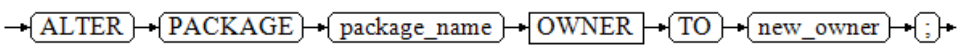
Precautions

- Currently, only users with the ALTER PACKAGE OWNER permission can run this command. By default, system administrators have the permission. The restrictions are as follows:
  - The current user must be the owner of the package or the system administrator and a member of the new owner role.
- The **plpgsql_dependency** parameter must be set for package recompilation.
- When separation of duties is enabled, to change the owner of a package, users must have the user group permission, even a system administrator. The owner of a package of the DEFINER type cannot be changed.
- Only the initial user can change the owner of a package to the initial user.
- When separation of duties is disabled, only a system administrator or a higher-level user can change the package owner, but the owner cannot be changed to an O&M administrator.
- The system administrator is not allowed to change the owner of a package of the DEFINER type to the initial user or O&M administrator.

Syntax

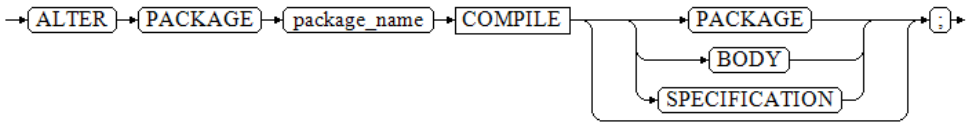
- Change the owner of a package.

```
ALTER PACKAGE package_name OWNER TO new_owner;
```



- Recompile the package.

```
ALTER PACKAGE package_name COMPILE [PACKAGE | BODY | SPECIFICATION];
```



Parameters

- **package_name**

  Specifies the name of the package to be modified.

Value range: an existing package name. Only one package can be modified at a time.

- **new_owner**

  Specifies the new owner of a package. To change the owner of a package, the new owner must have the CREATE permission on the schema to which the package belongs.

  Value range: an existing user role.

Example

- Change the owner of a package.

```
-- Create a package.
gaussdb=# CREATE OR REPLACE PACKAGE test_pkg AS
    pkg_var int := 1;
    PROCEDURE test_pkg_proc(var int);
END test_pkg;
/
gaussdb=# CREATE OR REPLACE PACKAGE BODY test_pkg AS
    PROCEDURE test_pkg_proc(var int) AS
    BEGIN
        pkg_var := 1;
    END;
END test_pkg;
/

-- Create a user.
gaussdb=# CREATE ROLE test PASSWORD '********';

-- Change the owner of a package.
gaussdb=# ALTER PACKAGE test_pkg OWNER TO test;

-- Query the owner of test_pkg.
gaussdb=# SELECT t1.pkgname,t2.rolname FROM gs_package t1, pg_roles t2 WHERE t1.pkgname =
'test_pkg' AND t1.pkgowner = t2.oid;
 pkgname  | rolname
----------+---------
 test_pkg | test
(1 row)
```

- Recompile the package.

```
-- Enable the dependency function.
gaussdb=# SET behavior_compat_options ='plpgsql_dependency';
-- Recompile the package.
gaussdb=# ALTER PACKAGE test_pkg COMPILE;

-- Delete.
gaussdb=# DROP PACKAGE test_pkg;
gaussdb=# DROP ROLE test;
-- Disable the dependent function.
gaussdb=# RESET behavior_compat_options;
```

Helpful Links

CREATE PACKAGE and **DROP PACKAGE**

7.12.6.21 ALTER PROCEDURE

Description

Modifies the attributes of a user-defined stored procedure or recompiles a stored procedure.

Precautions

- Only the owner of a stored procedure or a user granted with the ALTER permission can run the **ALTER PROCEDURE** command. The system administrator has this permission by default. The following are permission constraints depending on the attributes to be modified:
  - If a stored procedure involves operations on temporary tables, ALTER PROCEDURE cannot be used.
  - To modify the owner or schema of a stored procedure, you must be the owner of the stored procedure or system administrator and a member of the new owner role.
  - Only the system administrator and initial user can change the schema of a stored procedure to a public schema.
- The **plpgsql_dependency** parameter must be set for stored procedure compilation.
- Only the initial user or the user who creates the stored procedure can modify the stored procedure to a stored procedure that has the definer permission.
- When separation of duties is enabled, no role is allowed to modify the owner of a stored procedure with the definer permission.
- When separation of duties is disabled, only the initial user and system administrator can change the owner of a stored procedure with the definer permission. However, the stored procedure owner cannot be changed to an O&M administrator.
- Only the initial user can change the owner of a stored procedure to the initial user.

Syntax

- Modify the additional parameters of a user-defined stored procedure.

```
ALTER PROCEDURE procedure_name ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    action [ ... ] [ RESTRICT ];
```

  The syntax of the ACTION clause is as follows:

```
{CALLED ON NULL INPUT  | STRICT}
 | {IMMUTABLE | STABLE | VOLATILE}
 | {SHIPPABLE | NOT SHIPPABLE}
 | {NOT FENCED | FENCED}
 | [ NOT ] LEAKPROOF
 | { [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER }
 | AUTHID { DEFINER | CURRENT_USER }
 | COST execution_cost
 | ROWS result_rows
 | SET configuration_parameter { { TO | = } { value | DEFAULT }| FROM CURRENT}
 | RESET {configuration_parameter | ALL}
```

- Modify the name of a user-defined stored procedure.

```
ALTER PROCEDURE proname ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    RENAME TO new_name;
```

- Modify the owner of a user-defined stored procedure.

```
ALTER PROCEDURE proname ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    OWNER TO new_owner;
```

- Modify the schema of a user-defined stored procedure.

```
ALTER PROCEDURE proname ( [ { [ argname ] [ argmode ] argtype} [, ...] ] )
    SET SCHEMA new_schema;
```

- Recompile a stored procedure.

```
ALTER PROCEDURE procedure_name COMPILE;
```


Parameters

- **procedure_name**

  Specifies the name of the stored procedure to be modified.

  Value range: an existing stored procedure name

- **argmode**

  Specifies whether a parameter is an input or output parameter.

  Value range: **IN**, **OUT**, **INOUT**, and **VARIADIC**

- **argname**

  Specifies the parameter name.

  Value range: a string. It must comply with the **naming convention**.

- **argtype**

  Specifies the type of the stored procedure parameter.

- **CALLED ON NULL INPUT**

  Declares that some parameters of the stored procedure can be called in normal mode if the parameter values are null. Omitting this parameter is the same as specifying it.

- **IMMUTABLE**

  Specifies that the stored procedure always returns the same result if the parameter values are the same.

- **STABLE**

  Specifies that the stored procedure cannot modify the database, and that within a single table scan it will consistently return the same result for the same parameter value, but its result varies by SQL statements.

- **VOLATILE**

  Specifies that the stored procedure value can change in a single table scan and no optimization is performed.

- **LEAKPROOF**

  Specifies that the stored procedure has no side effect and the parameter contains only the return value. **LEAKPROOF** can be set only by the system administrator.

- **EXTERNAL**

  (Optional) The purpose is to be compatible with SQL. This feature applies to all functions, not only external functions.

- **SECURITY INVOKER**

  AUTHID CURRENT_USER

  Specifies that the stored procedure will be executed with the permissions of the user who calls it. Omitting this parameter is the same as specifying it.

  SECURITY INVOKER and **AUTHID CURRENT_USER** have the same functions.

- **SECURITY DEFINER**

  AUTHID DEFINER

  Specifies that the stored procedure will be executed with the permissions of the user who created it.

  AUTHID DEFINER and **SECURITY DEFINER** have the same functions.

- **COST execution_cost**

  Estimates the execution cost of the stored procedure.

  The unit of **execution_cost** is **cpu_operator_cost**.

  Value range: a positive integer

- **ROWS result_rows**

  Estimates the number of rows returned by the stored procedure. This is only allowed when the stored procedure is declared to return a set.

  Value range: a positive number. The default value is **1000**.

- **configuration_parameter**
  - **value**

    Sets a specified database session parameter to a specified value. If the value is **DEFAULT** or **RESET**, the default setting is used in the new session. **OFF** disables the setting.

    Value range: a string.

    - **DEFAULT**
    - **OFF**
    - **RESET**
    - User-specified value: The value must meet the restriction of the modified parameter.

  - **FROM CURRENT**

    Uses the value of **configuration_parameter** of the current session.

- **new_name**

  Specifies the new name of the stored procedure. To change the schema of a stored procedure, you must have the CREATE permission on the new schema.

  Value range: a string. It must comply with the **naming convention**.

- **new_owner**

  Specifies the new owner of the stored procedure. To change the owner of a stored procedure, the new owner must have the CREATE permission on the schema to which the stored procedure belongs.

  Value range: an existing user role

- **new_schema**

  Specifies the new schema of the stored procedure.

  Value range: an existing schema

Examples

See **Examples** in section "CREATE PROCEDURE."

Recompilation examples:

```
-- Enable the dependency function.
gaussdb=# SET behavior_compat_options ='plpgsql_dependency';

-- Create a stored procedure.
gaussdb=# CREATE OR REPLACE PROCEDURE test_proc(a int)
IS
```



```
    proc_var int;
BEGIN
    proc_var := a;
END;
/

-- Recompile a stored procedure with the stored procedure name.
gaussdb=# ALTER PROCEDURE test_proc COMPILE;

-- Recompile a stored procedure with a signed int type stored procedure.
gaussdb=# ALTER PROCEDURE test_proc(int) COMPILE;

-- Drop the stored procedure.
gaussdb=# DROP PROCEDURE test_proc;
```

Helpful Links

CREATE PROCEDURE and **DROP PROCEDURE**

7.12.6.22 ALTER RESOURCE LABEL

Description

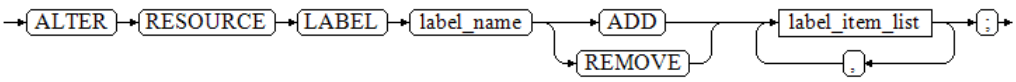
ALTER RESOURCE LABEL modifies a resource label.

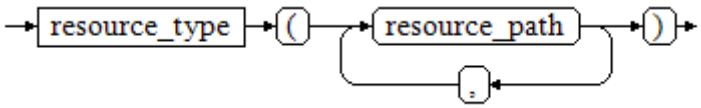
Precautions

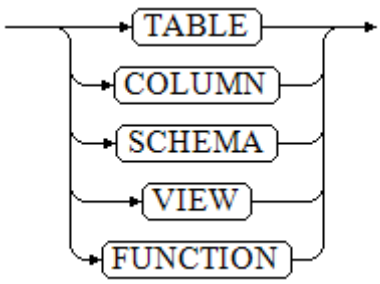
Only users with the poladmin or sysadmin permission, or the initial user can perform this operation.

Syntax

```
ALTER RESOURCE LABEL label_name {ADD|REMOVE}
  label_item_list[, ...];
```



- label_item_list
  ```
  resource_type(resource_path[, ...])
  ```
  

- resource_type
  ```
  TABLE | COLUMN | SCHEMA | VIEW | FUNCTION
  ```
  

Parameters

- **label_name**

  Specifies the resource label name.

  Value range: a string. It must comply with the **naming convention**.

- **resource_type**

  Specifies the type of database resources to be labeled.

- **resource_path**

  Specifies the path of database resources.

Examples

```
-- Create basic table table_for_label.
gaussdb=# CREATE TABLE table_for_label(col1 int, col2 text);

-- Create resource label table_label.
gaussdb=# CREATE RESOURCE LABEL table_label ADD COLUMN(table_for_label.col1);

-- Attach resource label table_label to col2.
gaussdb=# ALTER RESOURCE LABEL table_label ADD COLUMN(table_for_label.col2);

--  Remove an item from table_label.
gaussdb=# ALTER RESOURCE LABEL table_label REMOVE COLUMN(table_for_label.col1);

-- Delete the resource label table_label.
gaussdb=# DROP RESOURCE LABEL table_label;

-- Delete the base table table_for_label.
gaussdb=# DROP TABLE table_for_label;
```

Helpful Links

CREATE RESOURCE LABEL and **DROP RESOURCE LABEL**

7.12.6.23 ALTER RESOURCE POOL

Description

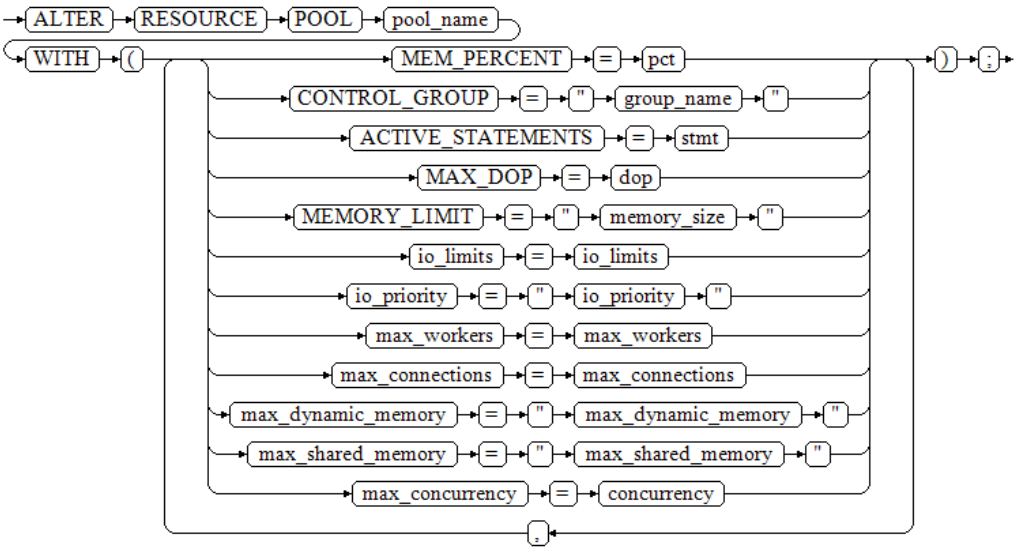
Modifies the Cgroup of a resource pool.

Precautions

Only users with the sysadmin permission, or the initial user can perform this operation.

Syntax

```
ALTER RESOURCE POOL pool_name
    WITH ({MEM_PERCENT= pct | CONTROL_GROUP="group_name" | ACTIVE_STATEMENTS=stmt |
MAX_DOP = dop | MEMORY_LIMIT="memory_size" | io_limits=io_limits | io_priority="io_priority" |
max_workers=max_workers | max_connections=max_connections |
max_dynamic_memory="max_dynamic_memory"| max_shared_memory="max_shared_memory" |
max_concurrency=concurrency}[, ... ]);
```

Parameters

- **pool_name**

  Specifies the name of a resource pool.

  The resource pool must already exist.

  Value range: a string. It must comply with the **naming convention**.

- **group_name**

  Specifies the name of a Cgroup.

  NOTE

  - You can use either double quotation marks ("") or single quotation marks (') in the syntax when setting the name of a Cgroup.
  - The value of **group_name** is case-sensitive.
  - If an administrator specifies a Workload Cgroup under Class, for example, **control_group** set to **class1:workload1**, the resource pool will be associated with the **workload1** Cgroup under **class1**. The level of **Workload** can also be specified. For example, **control_group** is set to **class1:workload1:1**.
  - If a database user specifies the Timeshare string (**Rush**, **High**, **Medium**, or **Low**) in the syntax, for example, **control_group** is set to **High**, the resource pool will be associated with the **High** Timeshare Cgroup under **DefaultClass**.

  Value range: an existing Cgroup.

- **stmt**

  Specifies the maximum number of statements that can be concurrently executed in a resource pool. The value **–1** indicates that the number of concurrent statements is not limited.

  Value range: numeric data ranging from –1 to 2147483647

- **dop**

  Specifies the maximum statement concurrency degree for a resource pool, equivalent to the number of threads that can be created for executing a statement.

  Value range: numeric data ranging from 1 to 2147483647

- **memory_size**

  Specifies the maximum memory size of a resource pool.

  Value range: a string of 1 KB to 2047 GB case-sensitive characters.

- **mem_percent**

  Specifies the proportion of available resource pool memory to the total memory or group user memory.

  In multi-tenant scenarios, **mem_percent** of group users or service users ranges from 1 to 100. The default value is **20**.

  In common scenarios, **mem_percent** of common users is an integer ranging from 0 to 100. The default value is **0**.

  NOTE

  When both of **mem_percent** and **memory_limit** are specified, only **mem_percent** takes effect.

- **io_limits**

  Specifies the upper limit of IOPS in a resource pool. The value **0** indicates no limit.

  It is counted by 10 thousands per second.

  Value range: numeric data ranging from 0 to 2147483647

- **io_priority**

  Specifies the I/O priority set for jobs that consume many I/O resources. It takes effect when the I/O usage reaches 90%.

  There are three priorities: **Low**, **Medium**, and **High**. If you do not want to control I/O resources, set this parameter to **None**, which is the default value.

  Value range: enumerated type. The options are **None**, **Low**, **Medium**, and **High**.

  NOTE

  The settings of **io_limits** and **io_priority** are valid only for complex jobs, such as batch import (using INSERT INTO SELECT, COPY FROM, or CREATE TABLE AS), complex queries involving over 500 MB data on each DN, and VACUUM FULL.

- **max_workers**

  Concurrency in a table during data redistribution. This is used only for scaling.

- **max_connections**

  Maximum number of connections that can be used by a resource pool.

  NOTE

  The total maximum number of connections in all resource pools cannot exceed the maximum number of connections specified by **max_connections** of the entire GaussDB process.

- **max_dynamic_memory**

  Specifies the maximum dynamic memory that can be used by a resource pool.

- **max_shared_memory**

  Specifies the maximum shared memory that can be used by a resource pool.

- **max_concurrency**

Specifies the maximum concurrent requests that can be used by a resource pool.

Examples

The following example assumes that the user has created the **class1** Cgroup and three Workload Cgroups under **class1**: **Low**, **wg1**, and **wg2**. Contact the administrator to create a Cgroup.

```
-- Create a resource pool.
gaussdb=# CREATE RESOURCE POOL pool1;

-- Specify the High Timeshare Workload Cgroup under the DefaultClass Cgroup.
gaussdb=# ALTER RESOURCE POOL pool1 WITH (CONTROL_GROUP="High");

-- Specify the Low Timeshare Workload Cgroup under the class1 Cgroup.
gaussdb=# ALTER RESOURCE POOL pool1 WITH (CONTROL_GROUP="class1:Low");

-- Specify the wg1 Workload Cgroup under the class1 Cgroup.
gaussdb=# ALTER RESOURCE POOL pool1 WITH (CONTROL_GROUP="class1:wg1");

-- Specify the wg2 Workload Cgroup under the class1 Cgroup.
gaussdb=# ALTER RESOURCE POOL pool1 WITH (CONTROL_GROUP="class1:wg2:3");

-- Delete the resource pool pool1.
gaussdb=# DROP RESOURCE POOL pool1;
```

Helpful Links

CREATE RESOURCE POOL and **DROP RESOURCE POOL**

7.12.6.24 ALTER ROLE

Description

Modifies role attributes.

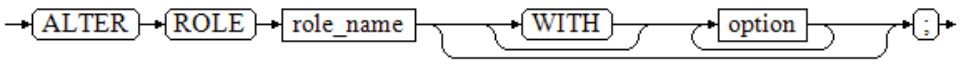
Precautions

None

Syntax

- Modify the permissions of a role.

```
ALTER ROLE role_name [ [ WITH ] option [ ... ] ];
```



  - The **option** clause for granting permissions is as follows:

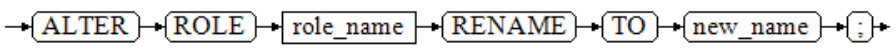
```
{CREATEDB | NOCREATEDB}
    | {CREATEROLE | NOCREATEROLE}
    | {INHERIT | NOINHERIT}
    | {AUDITADMIN | NOAUDITADMIN}
    | {SYSADMIN | NOSYSADMIN}
    | {MONADMIN | NOMONADMIN}
    | {OPRADMIN | NOOPRADMIN}
    | {POLADMIN | NOPOLADMIN}
    | {USEFT | NOUSEFT}
    | {LOGIN | NOLOGIN}
    | {REPLICATION | NOREPLICATION}
```



```
    | {PERSISTENCE | NOPERSISTENCE}
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD { 'password' [EXPIRED] | DISABLE | EXPIRED }
    | [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY { 'password' [ REPLACE 'old_password' |
EXPIRED ] | DISABLE }
    | VALID BEGIN 'timestamp'
    | VALID UNTIL 'timestamp'
    | RESOURCE POOL 'respool'
    | USER GROUP 'groupuser'
    | PERM SPACE 'spacelimit'
    | TEMP SPACE 'tmpspacelimit'
    | SPILL SPACE 'spillspacelimit'
    | NODE GROUP logic_cluster_name
    | ACCOUNT { LOCK | UNLOCK }
    | PGUSER
```

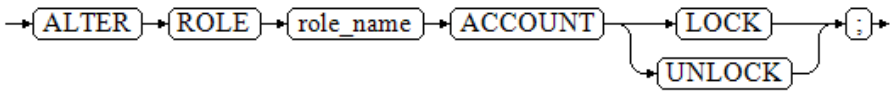
- Rename a role.

```
ALTER ROLE role_name
    RENAME TO new_name;
```



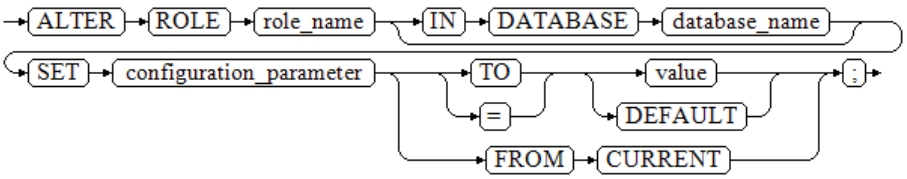
- Lock or unlock.

```
ALTER ROLE role_name
    ACCOUNT { LOCK | UNLOCK };
```



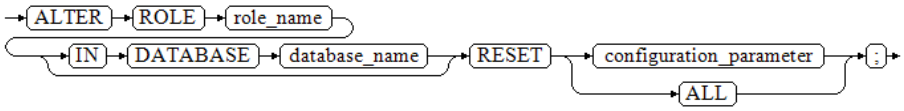
- Set parameters for a role.

```
ALTER ROLE role_name [ IN DATABASE database_name ]
    SET configuration_parameter {{ TO | = } { value | DEFAULT } | FROM CURRENT};
```



- Reset parameters for a role.

```
ALTER ROLE role_name
    [ IN DATABASE database_name ] RESET {configuration_parameter|ALL};
```



Parameters

- **role_name**

  Specifies a role name.

  Value range: an existing role name. If a role name contains uppercase letters, enclose the name with double quotation marks ("").

- **IN DATABASE database_name**

  Modifies the parameters of a role in a specified database.

- **SET configuration_parameter {{ TO | = } { value | DEFAULT } | FROM CURRENT}**

  Sets parameters for a role. Session parameters modified by ALTER ROLE apply to a specified role and take effect in the next session triggered by the role.

Value range:

For details about the values of **configuration_parameter** and **value**, see **SET**.

DEFAULT: clears the value of **configuration_parameter**. **configuration_parameter** will inherit the default value of the new session generated for the role.

FROM CURRENT: uses the value of **configuration_parameter** of the current session.

- **RESET {configuration_parameter|ALL}**

  Clears the value of **configuration_parameter**. The statement has the same effect as that of **SET configuration_parameter TO DEFAULT**.

  Value range: **ALL** indicates that the values of all parameters are cleared.

- **ACCOUNT LOCK | ACCOUNT UNLOCK**
  - **ACCOUNT LOCK**: locks an account to forbid login to databases.
  - **ACCOUNT UNLOCK**: unlocks an account to allow login to databases.
- **PGUSER**

  In the current version, the **PGUSER** attribute of a role cannot be modified.

- **{PASSWORD|IDENTIFIED BY} 'password'**

  Resets or changes the user password. Except the initial user, other administrators and common users need to enter the correct old password when changing their own passwords. Only the initial user, the system administrator (sysadmin) when separation of duties is disabled, or users who have the permission (CREATEROLE) to create users can reset the password of a common user without entering the old password. The initial user can reset passwords of system administrators. A system administrator cannot reset passwords of other system administrators. The user password should be enclosed by single quotation marks.

- **EXPIRED**

  Invalidates the password. Only the initial user, the system administrator (sysadmin), or user who has the permission to create a user (CREATEROLE) can invalidate the user password. A system administrator can invalidate the password of itself or other system administrators only when separation of duties is disabled. The password of the initial user cannot be invalidated.

  The user whose password is invalid can log in to the database but cannot perform the query operation. The query operation can be performed only after the password is changed or the administrator resets the password.

For details about other parameters, see **Parameters** in "CREATE ROLE."

Examples

```
-- Create role test_role.
gaussdb=# CREATE ROLE test_role PASSWORD '********';

-- Set role test_role to log in to the database.
gaussdb=# ALTER ROLE test_role WITH LOGIN;

-- Lock role test_role.
gaussdb=# ALTER ROLE test_role ACCOUNT LOCK;

-- Unlock a locked role.
gaussdb=# ALTER ROLE test_role ACCOUNT UNLOCK;
```



```
-- Change the password of role test_role.
gaussdb=# ALTER ROLE test_role PASSWORD '********';

-- Rename role test_role to test_role2.
gaussdb=# ALTER ROLE test_role RENAME TO test_role2;

-- Change role test_role2 to the system administrator.
gaussdb=# ALTER ROLE test_role2 SYSADMIN;

-- Delete.
gaussdb=# DROP ROLE test_role2;
```

Helpful Links

CREATE ROLE, **DROP ROLE**, and **SET ROLE**

7.12.6.25 ALTER ROW LEVEL SECURITY POLICY

Description

Modifies an existing row-level security policy, including the policy name and the users and expressions affected by the policy.

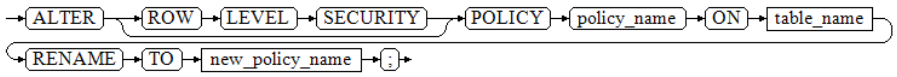
Precautions

Only the table owner or a system administrator can perform this operation.

Syntax

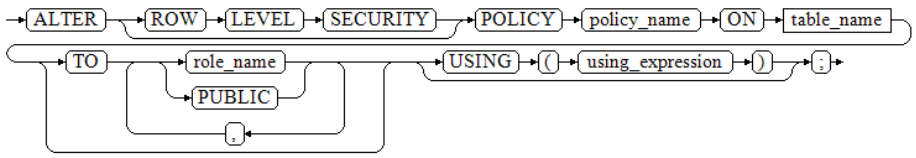
- Change the name of an existing row-level security policy.

```
ALTER [ ROW LEVEL SECURITY ] POLICY policy_name ON table_name RENAME TO new_policy_name;
```



- Change the specified user and policy expression of an existing row-level security policy.

```
ALTER [ ROW LEVEL SECURITY ] POLICY policy_name ON table_name
    [ TO { role_name | PUBLIC } [, ...] ]
    [ USING ( using_expression ) ];
```



Parameters

- **policy_name**

  Specifies the name of a row-level security policy.

- **table_name**

  Specifies the name of a table to which a row-level security policy is applied.

- **new_policy_name**

  Specifies the new name of a row-level security policy.

- **role_name**

Specifies names of users affected by a row-level security policy. **PUBLIC** indicates that the row-level security policy will affect all users.

- **using_expression**

  Specifies a row-level security policy, which is similar to a Boolean expression in the WHERE clause.

Examples

```
-- Create the data table all_data.
gaussdb=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));

-- Create a row-level security policy to specify that the current user can view only their own data.
gaussdb=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role = CURRENT_USER);
gaussdb=# \d+ all_data
                        Table "public.all_data"
 Column |          Type          | Modifiers | Storage  | Stats target | Description
--------+------------------------+-----------+----------+--------------+-------------
 id     | integer                |           | plain    |              |
 role   | character varying(100) |           | extended |              |
 data   | character varying(100) |           | extended |              |
Row Level Security Policies:
    POLICY "all_data_rls" FOR ALL
      TO public
      USING (((role)::name = "current_user"()))
Has OIDs: no
Options: orientation=row, compression=no

-- Create users alice and bob.
gaussdb=# CREATE ROLE alice WITH PASSWORD "********";
gaussdb=# CREATE ROLE bob WITH PASSWORD "********";

-- Change the name of the all_data_rls policy.
gaussdb=# ALTER ROW LEVEL SECURITY POLICY all_data_rls ON all_data RENAME TO all_data_new_rls;

-- Change the users affected by the row-level security policy.
gaussdb=# ALTER ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data TO alice, bob;
gaussdb=# \d+ all_data
                        Table "public.all_data"
 Column |          Type          | Modifiers | Storage  | Stats target | Description
--------+------------------------+-----------+----------+--------------+-------------
 id     | integer                |           | plain    |              |
 role   | character varying(100) |           | extended |              |
 data   | character varying(100) |           | extended |              |
Row Level Security Policies:
    POLICY "all_data_new_rls" FOR ALL
      TO alice,bob
      USING (((role)::name = "current_user"()))
Has OIDs: no
Options: orientation=row, compression=no, enable_rowsecurity=true

-- Modify the expression defined for the row-level security policy.
gaussdb=# ALTER ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data USING (id > 100 AND role =
current_user);
gaussdb=# \d+ all_data
                        Table "public.all_data"
 Column |          Type          | Modifiers | Storage  | Stats target | Description
--------+------------------------+-----------+----------+--------------+-------------
 id     | integer                |           | plain    |              |
 role   | character varying(100) |           | extended |              |
 data   | character varying(100) |           | extended |              |
Row Level Security Policies:
    POLICY "all_data_new_rls" FOR ALL
      TO alice,bob
      USING (((id > 100) AND ((role)::name = "current_user"())))
Has OIDs: no
Options: orientation=row, compression=no, enable_rowsecurity=true
```



```
-- Delete the policy.
gaussdb=# DROP ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data;

-- Delete users alice and bob.
gaussdb=# DROP ROLE alice, bob;

-- Delete the all_data table.
gaussdb=# DROP TABLE all_data;
```

Helpful Links

CREATE ROW LEVEL SECURITY POLICY and **DROP ROW LEVEL SECURITY POLICY**

7.12.6.26 ALTER SCHEMA

Description

Modifies schema attributes.

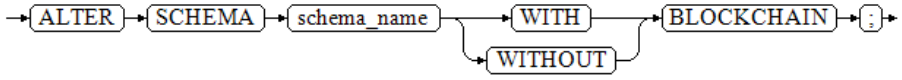
Precautions

- Only the owner of a schema or users granted with the ALTER permission on the schema can run the **ALTER SCHEMA** command. When separation of duties is disabled, system administrators have this permission by default. To change the owner of a schema, you must be the owner of the schema or system administrator and a member of the new owner role.
- For system schemas other than public, such as pg_catalog and sys, only the initial user is allowed to change the owner of a schema. Changing the names of the built-in system schemas may make some functions unavailable or even affect the normal running of the database. By default, the names of the built-in system schemas cannot be changed. To ensure forward compatibility, you can change the names of the built-in system schemas only when the system is being started or upgraded or when **allow_system_table_mods** is set to **on**.
- Except the initial user, other users cannot change the owner of a schema to an O&M administrator.

Syntax

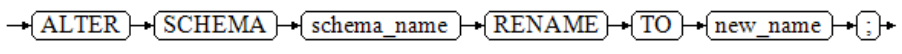
- Alter the tamper-proof attribute of a schema.

```
ALTER SCHEMA schema_name { WITH | WITHOUT } BLOCKCHAIN;
```



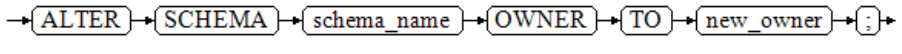
- Rename a schema.

```
ALTER SCHEMA schema_name
  RENAME TO new_name;
```



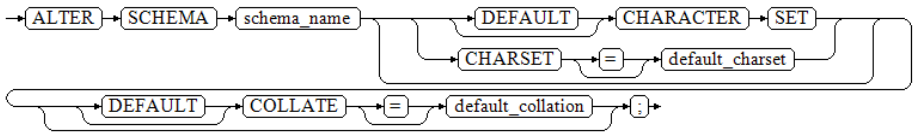
- Change the owner of a schema.

```
ALTER SCHEMA schema_name
  OWNER TO new_owner;
```



- Modify the default character set and collation of the schema.

```
ALTER SCHEMA schema_name
    [ [DEFAULT] CHARACTER SET | CHARSET [ = ] default_charset ] [ [DEFAULT] COLLATE [ = ]
default_collation ];
```



Parameters

- **schema_name**

  Specifies the name of an existing schema.

  Value range: an existing schema name.

- **RENAME TO new_name**

  Renames a schema. If a non-administrator user wants to change the schema name, the user must have the CREATE permission on the database.

  new_name: new name of the schema.

  NOTICE

  - The schema name must be unique in the current database.
  - The schema name cannot be the same as the initial username of the current database.
  - The schema name cannot start with **pg_**.
  - The schema name cannot start with **gs_role_**.

  Value range: a string. It must comply with the **naming convention**.

- **OWNER TO new_owner**

  Changes the owner of a schema. To do this as a non-administrator, you must be a direct or indirect member of the new owner role, and that role must have the CREATE permission on the database.

  new_owner: new owner of the schema.

  Value range: an existing username or role name.

- **{ WITH | WITHOUT } BLOCKCHAIN**

  Alters the tamper-proof attribute of a schema by using WITH. Common row-store tables with the tamper-proof attribute are tamper-proof history tables, excluding foreign tables, temporary tables, and system catalogs. The tamper-proof attribute can be altered only when no table is contained in the schema. The tamper-proof attribute of the temporary table schema, the TOAST table schema, dbe_perf schema, and blockchain schema cannot be modified. This syntax can be used to convert between normal and tamper-proof modes only if the schema does not contain any tables.

  NOTE

  To change a common schema to a tamper-proof schema, set the GUC parameter **enable_ledger** to **on**. The default value is **off**, and the level is SIGHUP.

- **default_charset**

  Modifies the default character set of a schema. If you specify a schema separately, the default collation of the schema is set to the default collation of the specified character set.

  This syntax is supported only when **sql_compatibility** is set to **'B'**. For details about the supported character sets, see **Table 7-220**.

- **default_collate**

  This command is used to change the default collation of a schema. If you specify a collation separately, the default character set of the schema is set to the character set corresponding to the specified collation.

  This syntax is supported only when **sql_compatibility** is set to **'B'**. For details about the supported collation, see **Table 7-220**.

Examples

- Alter the tamper-proof attribute of a schema.

  To alter the common mode to the tamper-proof mode, you need to set the GUC parameter **enable_ledger** to determine whether to enable the ledger database function. Contact the administrator for information about how to use parameters.

```
-- Create a schema test_schema1.
gaussdb=# CREATE SCHEMA test_schema1;

-- Change the schema test_schema1 to the tamper-proof mode.
gaussdb=# ALTER SCHEMA test_schema1 WITH BLOCKCHAIN;

-- Query mode information. The tamper-proof attribute is true.
gaussdb=# \dn+ test_schema1
                          List of schemas
     Name     | Owner | Access privileges | Description | WithBlockChain
--------------+-------+-------------------+-------------+----------------
 test_schema1 | omm   |                   |             | t
(1 row)
```

- Rename a schema.

```
-- Rename the schema test_schema1 to test_sch1.
gaussdb=# ALTER SCHEMA test_schema1 RENAME TO test_sch1;

-- Query schema information.
gaussdb=# \dn+ test*
                         List of schemas
   Name    | Owner | Access privileges | Description | WithBlockChain
-----------+-------+-------------------+-------------+----------------
 test_sch1 | omm   |                   |             | t
(1 row)
```

- Change the owner of a schema.

```
-- Create user test_user.
gaussdb=# CREATE ROLE test_user PASSWORD '********';

-- Change the owner of schema test_sch1 to test_user.
gaussdb=# ALTER SCHEMA test_sch1 OWNER TO test_user;

-- Query schema information.
gaussdb=# \dn+  test_sch1;
                         List of schemas
   Name    |   Owner   | Access privileges | Description | WithBlockChain
-----------+-----------+-------------------+-------------+----------------
 test_sch1 | test_user |                   |             | t
(1 row)

-- Delete.
```



```
gaussdb=# DROP SCHEMA test_sch1;
gaussdb=# DROP ROLE test_user;
```

- Modify the default character set and collation.

  This syntax is supported only when **sql_compatibility** is set to **'B'**.

  Except the binary character set and collation, only the character set that is the same as the database encoding can be specified.

```
-- Create and switch to the test database.
gaussdb=# CREATE DATABASE test1 WITH DBCOMPATIBILITY = 'B' ENCODING = 'UTF8' LC_COLLATE
= 'zh_CN.utf8' LC_CTYPE = 'zh_CN.utf8';

gaussdb=# \c test1

-- Create a schema test_sch2.
test1=# CREATE SCHEMA test_sch2;

-- Change the default character set to utf8mb4 and the default collation of characters to
utf8mb4_bin.
test1=# ALTER SCHEMA test_sch2 CHARACTER SET utf8mb4 COLLATE utf8mb4_bin;

-- Delete.
test1=# DROP SCHEMA test_sch2;

-- Switch to the default database. Change the database name based on actual situation.
test1=# \c postgres
gaussdb=# DROP DATABASE test1;
```

Helpful Links

CREATE SCHEMA and **DROP SCHEMA**

7.12.6.27 ALTER SEQUENCE

Description

Modifies the parameters of an existing sequence.

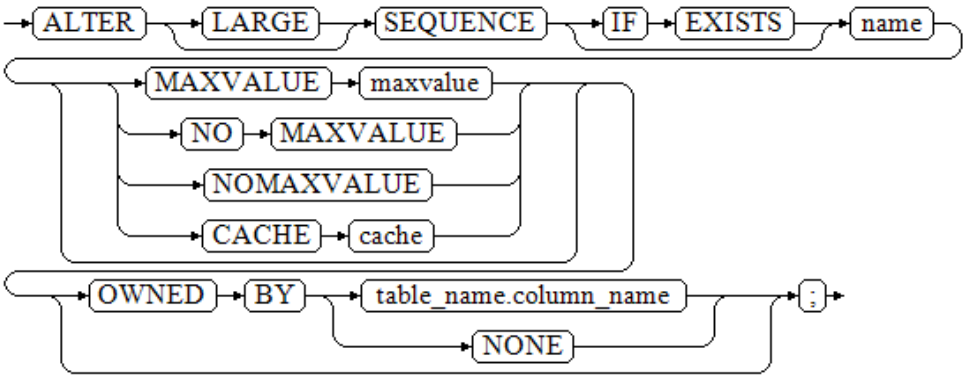
Precautions

- Only the owner of a sequence, a user granted the ALTER permission on a sequence, or a user granted the ALTER ANY SEQUENCE permission on a sequence can run the **ALTER SEQUENCE** command. When separation of duties is disabled, a system administrator has this permission by default. To modify a sequence owner, you must be the sequence owner or system administrator and a member of the new owner role.
- In the current version, you can modify only the owner, owning column, maximum value, and cache value. To modify other parameters, delete the sequence and create it again. Then, use the Setval function to restore parameter values.
- ALTER SEQUENCE MAXVALUE cannot be used in transactions, functions, and stored procedures.
- After the maximum value of a sequence is changed, the cache of the sequence in all sessions is cleared.
- If the LARGE identifier is used when a sequence is created, the LARGE identifier must be used when the sequence is altered.
- The ALTER SEQUENCE statement blocks the calling of nextval, setval, currval, and lastval.

Syntax

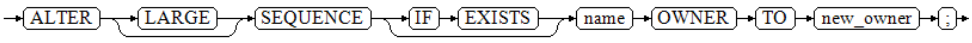
- Change the maximum sequence value, owning column, and cache value.

```
ALTER [ LARGE ] SEQUENCE [ IF EXISTS ] name
    [MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE | CACHE cache]
    [ OWNED BY { table_name.column_name | NONE } ] ;
```



- Change the owner of a sequence.

```
ALTER [ LARGE ] SEQUENCE [ IF EXISTS ] name OWNER TO new_owner;
```



Parameters

- **name**

  Specifies the name of the sequence to be modified.

- **IF EXISTS**

  This option is used when the sequence does not exist. ERROR is not displayed. Instead, a NOTICE message is returned.

- **MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE**

  Specifies the maximum value of the sequence. The new maximum value must be greater than **last_value**. If not specified, the old maximum value is retained.

  Value range: (last_value, 2^{63} – 1]. If LARGE is used, the range is (last_value, 2^{127} – 1].

- **CACHE**

  Specifies the number of sequences stored in the memory for quick access purposes. If this parameter is not specified, the old cache value is retained.

  Value range: [1, 2^{63} – 1]. If LARGE is used, the range is [1, 2^{127} – 1].

- **OWNED BY**

  Associates a sequence with a specified column included in a table. In this way, the sequence will be deleted when you delete its associated column or the table where the column belongs to.

  If the sequence has been associated with another table before you use this option, the new association will overwrite the old one.

  The associated table and sequence must be owned by the same user and in the same schema.

  If OWNED BY NONE is used, all existing associations will be deleted.

- **new_owner**

  Specifies the username of the new owner of the sequence. To change the owner, you must also be a direct or indirect member of the new role, and this role must have the CREATE permission on the sequence's schema.

Examples

```
-- Create an ascending sequence named serial, which starts from 101.
gaussdb=# CREATE SEQUENCE serial START 101;

-- Create a table and specify default values for the sequence.
gaussdb=# CREATE TABLE t1(c1 bigint default nextval('serial'));

-- Change the owning column of serial to T1.C1.
gaussdb=# ALTER SEQUENCE serial OWNED BY t1.c1;

-- Delete a sequence and a table.
gaussdb=# DROP SEQUENCE serial CASCADE;
gaussdb=# DROP TABLE t1;
```

Helpful Links

CREATE SEQUENCE and **DROP SEQUENCE**

7.12.6.28 ALTER SERVER

Description

Adds, modifies, or deletes the parameters of an existing server. You can query existing servers from the pg_foreign_server system catalog.

Precautions

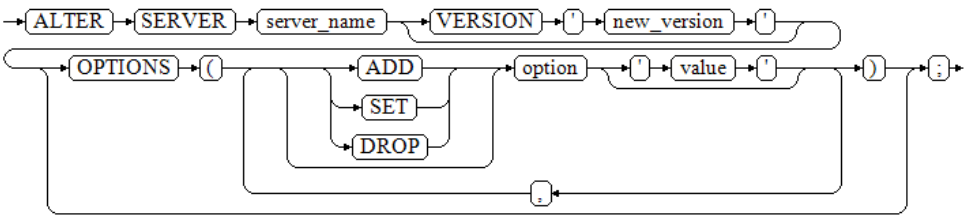
- Only the server owner or a user granted with the ALTER permission can run the **ALTER SERVER** command. By default, system administrators have the permission. To modify a server owner, you must be the server owner or system administrator and a member of the new owner role.
- When multi-layer quotation marks are used for sensitive columns (such as **password** and **secret_access_key**) in OPTIONS, the semantics is different from that in the scenario where quotation marks are not used. Therefore, sensitive columns are not identified for anonymization.

Syntax

- Change the parameters for a foreign server.

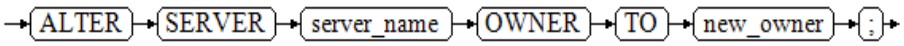
  ```
  ALTER SERVER server_name [ VERSION 'new_version' ]
      [ OPTIONS ( {[ ADD | SET | DROP ] option ['value']} [, ... ] ) ];
  ```

  In **OPTIONS**, **ADD**, **SET**, and **DROP** are operations to be performed. If these operations are not specified, ADD operations will be performed by default. **option** and **value** are the parameters of the corresponding operation.

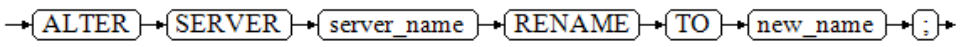
- Change the owner of a foreign server.

```
ALTER SERVER server_name
    OWNER TO new_owner;
```



- Change the name of a foreign server.

```
ALTER SERVER server_name
  RENAME TO new_name;
```



Parameters

- **server_name**

  Specifies the name of the server to be modified.

- **new_version**

  Specifies the new version of the server.

- **OPTIONS**

  Changes options of the server. ADD, SET, and DROP are operations to be performed. If the operation is not set explicitly, ADD is used. The option name must be unique, and the name and value are also validated with the foreign data wrapper library of the server.

  In addition to the connection parameters supported by libpq, the following parameters are provided:

  – **fdw_startup_cost**

    Estimates the startup time required for a foreign table scan. This value usually contains the time taken to establish a connection, analyze the request at the remote end, and generate a plan. The default value is **100**. The value is a real number greater than 0.

  – **fdw_tuple_cost**

    Specifies the additional consumption when each tuple is scanned on a remote server. The value specifies the extra consumption of data transmission between servers. The default value is **0.01**. The value is a real number greater than 0.

- **new_name**

  Specifies the new name of the server.

Examples

```
-- Create my_server.
gaussdb=# CREATE SERVER my_server FOREIGN DATA WRAPPER log_fdw;

-- Change the name of an external service.
gaussdb=# ALTER SERVER my_server RENAME TO my_server_1;

-- Delete my_server_1.
gaussdb=# DROP SERVER my_server_1;
```

Helpful Links

CREATE SERVER and **DROP SERVER**

7.12.6.29 ALTER SESSION

Description

ALTER SESSION defines or modifies the conditions or parameters that affect the current session. Modified session parameters are kept until the current session is disconnected.

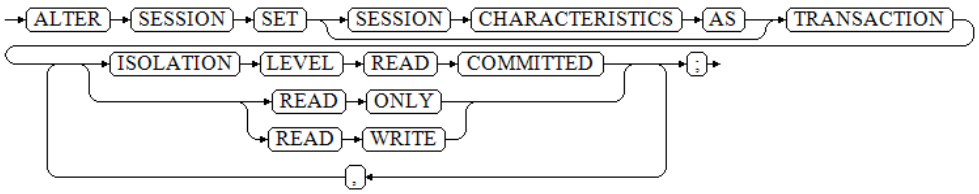
Precautions

- If the START TRANSACTION statement is not executed before the SET TRANSACTION statement, the transaction is ended instantly and the statement does not take effect.
- You can use the transaction_mode(s) method declared in the START TRANSACTION statement to avoid using the SET TRANSACTION statement. For details, see **START TRANSACTION**.

Syntax

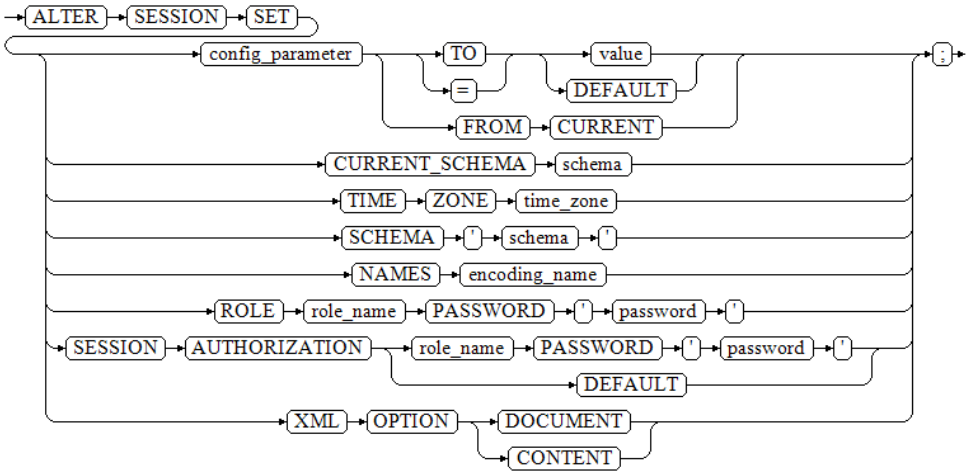
- Set transaction parameters of a session.

```
ALTER SESSION SET [ SESSION CHARACTERISTICS AS ] TRANSACTION
   { ISOLATION LEVEL { READ COMMITTED } | { READ ONLY  | READ WRITE } } [, ...] ;
```



- Set other GUC parameters of a session.

```
ALTER SESSION SET
   {{config_parameter { { TO  | =  }  { value | DEFAULT } | FROM CURRENT }}
     | CURRENT_SCHEMA schema
     | TIME ZONE time_zone
     | SCHEMA 'schema'
| NAMES encoding_name
     | ROLE role_name PASSWORD 'password'
     | SESSION AUTHORIZATION { role_name PASSWORD 'password' | DEFAULT }
     | XML OPTION { DOCUMENT | CONTENT }
   } ;
```



Parameters

- **config_parameter**

  Specifies the name of a configurable GUC parameter. You can use SHOW ALL to view available GUC parameters.

  - **value**

    Specifies the new value of **config_parameter**. This parameter can be specified as string constants, identifiers, numbers, or comma-separated lists of these. **DEFAULT** is used to set default values for parameters.

    - **DEFAULT**
    - **OFF**
    - **RESET**
    - User-specified value: The value must meet the restriction of the modified parameter.

  - **FROM CURRENT**

    Uses the value of **configuration_parameter** of the current session.

- **TIME ZONE timezone**

  Specifies the local time zone for the current session.

  Value range: a valid local time zone. The corresponding GUC parameter is **TimeZone**. The default value is **PRC**.

- **CURRENT_SCHEMA**

  schema

  Specifies the current schema.

  Value range: an existing schema name. If the schema name does not exist, the value of **CURRENT_SCHEMA** will be empty.

- **SCHEMA schema**

  Specifies the current schema. Here the schema is a string.

- **NAMES encoding_name**

  Specifies the client character encoding. This statement is equivalent to **set client_encoding to encoding_name**.

  Value range: a valid character encoding name. The GUC parameter corresponding to this option is **client_encoding**. The default encoding is **UTF8**.

- **role_name**

  Value range: a string. It must comply with the **naming convention**.

- **password**

  Specifies the password of a role. It must comply with the password convention.

- **SESSION AUTHORIZATION**

  Sets the session user identifier of the current session.

- **XML OPTION { DOCUMENT | CONTENT }**

  Specifies the XML parsing mode.

Value range: **CONTENT** (default) and **DOCUMENT**

Examples

- Set transaction parameters of a session.

  The keyword ALTER SESSION can be omitted in the example.

```
-- Start a transaction and set the transaction level.
gaussdb=# START TRANSACTION;
gaussdb=# ALTER SESSION SET TRANSACTION READ ONLY;
gaussdb=# END;
```

- Set other GUC parameters of a session.

  The keyword ALTER SESSION can be omitted in the example.

```
-- Create the ds schema.
gaussdb=# CREATE SCHEMA ds;

-- Set the search path of a schema.
gaussdb=# SET SEARCH_PATH TO ds, public;

-- Set the time/date type to the traditional postgres format (date before month).
gaussdb=# SET DATESTYLE TO postgres, dmy;

-- Set the character code of the current session to UTF8.
gaussdb=# ALTER SESSION SET NAMES 'UTF8';

-- Set the time zone to Berkeley of California.
gaussdb=# SET TIME ZONE 'PST8PDT';

-- Set the time zone to Italy.
gaussdb=# SET TIME ZONE 'Europe/Rome';

-- Set the current schema.
gaussdb=# ALTER SESSION SET CURRENT_SCHEMA TO tpcds;

-- Set XML OPTION to DOCUMENT.
gaussdb=# ALTER SESSION SET XML OPTION DOCUMENT;

-- Create the role joe, and set the session role to joe.
gaussdb=# CREATE ROLE joe WITH PASSWORD '********';
gaussdb=# ALTER SESSION SET SESSION AUTHORIZATION joe PASSWORD '********';

-- Switch to the default user.
gaussdb=> ALTER SESSION SET SESSION AUTHORIZATION default;

-- Delete the ds schema.
gaussdb=# DROP SCHEMA ds;

-- Delete joe.
gaussdb=# DROP ROLE joe;
```

Helpful Links

SET

7.12.6.30 ALTER SYNONYM

Description

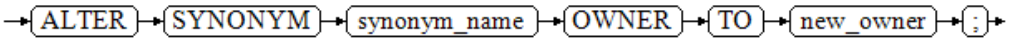
Changes the owner of a SYNONYM object.

Precautions

- Currently, only the owner of the SYNONYM object can be changed.
- Only a system administrator has the permission to modify the owner of the SYNONYM object. When separation of duties is enabled, system administrators do not have the permission to change the owner of the SYNONYM object by default.
- The new owner must have the CREATE permission on the schema where the SYNONYM object resides.
- PUBLIC synonyms cannot be modified.

Syntax

```
ALTER SYNONYM synonym_name
    OWNER TO new_owner;
```



Parameters

- **synonym_name**

  Specifies the name of the synonym to be modified, which can contain the schema name.

  Value range: a string. It must comply with the **naming convention**.

- **new_owner**

  Specifies the new owner of the SYNONYM object.

  Value range: a string. It must be a valid username.

Examples

```
-- Create a system administrator.
gaussdb=# CREATE USER sysadmin WITH SYSADMIN PASSWORD '********';

-- Switch the system administrator.
gaussdb=# \c - sysadmin

-- Create synonym t1.
gaussdb=# CREATE OR REPLACE SYNONYM t1 FOR ot.t1;

-- Create user u1.
gaussdb=# CREATE USER u1 PASSWORD '********';

-- Assign permissions to the new system administrator.
gaussdb=# GRANT ALL ON SCHEMA sysadmin TO u1;

-- Change the owner of synonym t1 to u1.
gaussdb=# ALTER SYNONYM t1 OWNER TO u1;

-- Delete synonym t1.
gaussdb=# DROP SYNONYM t1;

-- Revoke permissions from user u1.
gaussdb=# REVOKE ALL ON SCHEMA sysadmin FROM u1;

-- Delete user u1.
gaussdb=# DROP USER u1;

-- Switch to the initial user init_user. Replace init_user with the actual initial username.
gaussdb=# \c - init_user
```



```
-- Delete user sysadmin.
gaussdb=# DROP USER sysadmin;
```

Helpful Links

CREATE SYNONYM and **DROP SYNONYM**

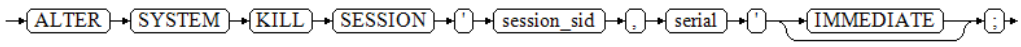
7.12.6.31 ALTER SYSTEM KILL SESSION

Description

ALTER SYSTEM KILL SESSION ends a session.

Syntax

```
ALTER SYSTEM KILL SESSION 'session_sid, serial' [ IMMEDIATE ];
```



Parameters

- **session_sid, serial**

  Specifies the SID and SERIAL of a session (To obtain the values, see the example.) You can use the pg_stat_activity system catalog to query the current active threads (see the examples). However, when you run the **ALTER SYSTEM KILL SESSION** command, the threads may have ended.

  Value range: SIDs and SERIALs of all sessions that can be queried from the system catalog dv_sessions.

- **IMMEDIATE**

  Specifies that a session will be ended instantly after the statement is executed.

Examples

```
-- Start two sessions, and create a table and start a transaction to insert data in the first session.
gaussdb=# CREATE TABLE tbl_test(id int);
gaussdb=# BEGIN;
gaussdb=# INSERT INTO tbl_test VALUES (1);

-- Query the session information in the second session. If the value of state is idle in transaction, the transaction is waiting to be committed.
gaussdb=# SELECT t1.datname,
       t1.usename,
       t1.pid,
       t2.serial#,
       t1.state
FROM pg_stat_activity t1,
     dv_sessions t2
WHERE t1.query LIKE 'INSERT INTO tbl_test%'
  AND t1.sessionid = t2.sid;
 datname  | usename |       pid       | serial# |        state
----------+---------+-----------------+---------+---------------------
 postgres | omm     | 139802072635136 |       0 | idle in transaction
(1 row)

-- End a session. If the IMMEDIATE parameter is not specified, the session and transactions in the session are forcibly ended.
```



```
gaussdb=# ALTER SYSTEM KILL SESSION '139802072635136,0';
 pg_terminate_backend
----------------------
 t
(1 row)

-- Perform the reconnection and query the tbl_test table. The transaction is forcibly ended and the data is
rolled back.
gaussdb=# SELECT * FROM tbl_test;
 id
----
(0 rows)

-- Drop the table.
gaussdb=# DROP TABLE tbl_test;
```

7.12.6.32 ALTER TABLE

Description

Modifies tables, including modifying table definitions, renaming tables, renaming specified columns in tables, renaming table constraints, setting table schemas, enabling or disabling row-level security policies, and adding or updating multiple columns.

Precautions

- The owner of a table, users granted with the ALTER permission on the table, or users granted with the ALTER ANY TABLE permission can run the **ALTER TABLE** command. The system administrator has the permission to run the command by default. To modify the owner or schema of a table, you must be the table owner or system administrator and a member of the new owner role.
- The tablespace of a partitioned table cannot be modified, but the tablespace of the partition can be modified.
- The storage parameter **ORIENTATION** cannot be modified.
- Currently, SET SCHEMA can only set schemas to user schemas. It cannot set a schema to a system internal schema.
- The partition key columns of a partitioned table cannot be changed or the character set cannot be converted.
- A column whose **DEFAULT** value contains the nextval() expression cannot be added.
- Row-level security cannot be enabled for foreign tables and temporary tables.
- When you delete a PRIMARY KEY constraint by constraint name, the NOT NULL constraint is not deleted. If necessary, manually delete the NOT NULL constraint.
- When JDBC is used, the **DEFAULT** value can be set through PrepareStatement.
- If you add a column using ADD COLUMN, all existing rows in the table are initialized to the column's default value (**NULL** if no **DEFAULT** value is specified).

  If no **DEFAULT** value is specified for the new column, **NULL** is used, and no full table update is triggered.

If the new column has the **DEFAULT** value, the column must meet all the following requirements. Otherwise, the entire table is updated, leading to additional overheads and affecting online services.

1. The data type is BOOL, BYTEA, TINYINT, SMALLINT, BIGINT, INTEGER, NUMERIC, FLOAT, DOUBLE PRECISION, CHAR, VARCHAR, TEXT, TIMESTAMPTZ, TIMESTAMP, DATE, TIME, TIMETZ, INTERVAL, SERIAL, BIGSERIAL, SMALLSERIAL, FLOATVECTOR, or BOOLVECTOR.

2. The length of the **DEFAULT** value of the added column cannot exceed 128 bytes.

3. The **DEFAULT** value of the added column does not contain the volatile function.

4. The **DEFAULT** value is required and cannot be **NULL**.

If you are not sure whether condition 3 is met, check whether the **provolatile** attribute of the function in the PG_RPOC system catalog is **v**.

5. If the PostgreSQL compatibility mode is enabled, the data type of the new column can be SERIAL, BIGSERIAL, or SMALLSERIAL. Only common tables are supported, and partitioned tables, temporary tables, unlogged permanent tables are not supported. The Astore and Ustore storage engines are supported.

- If **FIRST | AFTER column_name** is used to modify the character set of a column, add or modify a column, the update overhead of entire table is generated and affects online services.
- If you add a row-level expression when adding or changing an ILM policy for a data object, note that the row-level expression supports only the functions listed in the whitelist. For details about the whitelist function list, see **Row Expression Function Whitelist**.
- The number of table constraints cannot exceed 32,767.

Syntax

- Modify the definition of a table.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }
    action [, ... ];
ALTER TABLE [ IF EXISTS ] table_name
    ADD ( { column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint
[ ... ] ]} [, ...] );
ALTER TABLE [ IF EXISTS ] table_name
    MODIFY ( { column_name data_type | column_name [ CONSTRAINT constraint_name ] NOT NULL
[ ENABLE ] | column_name [ CONSTRAINT constraint_name ] NULL } [, ...] );
ALTER TABLE [ IF EXISTS ] table_name
    RENAME [TO | AS | = ] new_table_name;
RENAME {TABLE | TABLES} {table_name TO new_table_name} [, ...];
ALTER TABLE [ IF EXISTS ] { table_name  [*] | ONLY table_name | ONLY ( table_name  )}
    RENAME [ COLUMN ] column_name TO new_column_name;
ALTER TABLE [ IF EXISTS ] { table_name  [*] | ONLY table_name | ONLY ( table_name  )}
    RENAME CONSTRAINT constraint_name TO new_constraint_name;
ALTER TABLE [ IF EXISTS ] table_name
    SET SCHEMA new_schema;
ALTER TABLE [ IF EXISTS ] table_name GSIWAITALL;
```

The table operation **action** can be one of the following clauses:

```
column_clause
    | ADD table_constraint [ NOT VALID ]
    | ADD table_constraint_using_index
    | VALIDATE CONSTRAINT constraint_name
    | DROP CONSTRAINT [ IF EXISTS ]  constraint_name [ RESTRICT | CASCADE ]
    | DROP PRIMARY KEY
```



```
  | DROP FOREIGN KEY fk_symbol
  | CLUSTER ON index_name
  | SET WITHOUT CLUSTER
  | SET ( {storage_parameter = value} [, ... ] )
  | RESET ( storage_parameter [, ... ] )
  | OWNER TO new_owner
  | SET TABLESPACE new_tablespace
  | TO { GROUP groupname | NODE ( nodename [, ... ] ) }
  | ADD NODE ( nodename [, ... ] )
  | DELETE NODE ( nodename [, ... ] )
  | UPDATE SLICE LIKE table_name
  | DISABLE TRIGGER [ trigger_name | ALL | USER ]
  | ENABLE TRIGGER [ trigger_name | ALL | USER ]
  | ENABLE REPLICA TRIGGER trigger_name
  | ENABLE ALWAYS TRIGGER trigger_name
  | ENABLE ROW LEVEL SECURITY
  | DISABLE ROW LEVEL SECURITY
  | FORCE ROW LEVEL SECURITY
  | NO FORCE ROW LEVEL SECURITY
  | ENCRYPTION KEY ROTATION
  | REPLICA IDENTITY { DEFAULT | USING INDEX index_name | FULL | NOTHING }
  | AUTO_INCREMENT [ = ] value
  | COMMENT [ = ] 'string'
  | [ [ DEFAULT ] CHARACTER SET | CHARSET [ = ] default_charset ] [ [ DEFAULT ] COLLATE [ = ]
default_collation ]
  | CONVERT TO CHARACTER SET | CHARSET charset [ COLLATE collation ]
  | ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year }
OF { NO MODIFICATION } [ ON ( EXPR )]
  | [ MODIFY { PARTITION | SUBPARTITION }] ILM { ENABLE | DISABLE | DELETE } POLICY
policy_name
  | [ MODIFY { PARTITION | SUBPARTITION }] ILM { ENABLE_ALL | DISABLE_ALL | DELETE_ALL}
```


NOTE

- **ADD table_constraint [NOT VALID]**

  Adds a table constraint.

- **ADD table_constraint_using_index**

  Adds a primary key constraint or unique constraint to a table based on the existing unique index.

- **VALIDATE CONSTRAINT constraint_name**

  Validates a check-class constraint created with the **NOT VALID** option, and scans the entire table to ensure that all rows meet the constraint. Nothing happens if the constraint is already marked valid.

- **DROP CONSTRAINT [IF EXISTS] constraint_name [RESTRICT | CASCADE]**

  Deletes a table constraint.

- **DROP PRIMARY KEY**

  Deletes a primary key constraint from a table.

  This syntax is valid only when **sql_compatibility** is set to **'B'**.

- **DROP FOREIGN KEY fk_symbol**

  Deletes foreign key constraints from a table. **fk_symbol** indicates the name of the foreign key constraint to be deleted. This syntax is valid only when **sql_compatibility** is set to **'B'**.

- **CLUSTER ON index_name**

  Selects the default index for future CLUSTER operations. Actually, the table is not re-clustered.

- **SET WITHOUT CLUSTER**

  Deletes the most recently used CLUSTER index from the table. This affects future CLUSTER operations that do not specify an index.

- **SET ({storage_parameter = value} [, ...])**

  Changes one or more storage parameters for the table. If the value of **table_name** is an index name, **ACTIVE_PAGES** specifies the number of index pages, which may be less than the actual number of physical file pages and can be used for optimizer optimization. Currently, this parameter is valid only for the local index of the Ustore partitioned table and will be updated by VACUUM and ANALYZE (including AUTO VACUUM). You are advised not to manually set this parameter.

- **RESET (storage_parameter [, ...])**

  Resets one or more storage parameters to their defaults. As with SET, a table rewrite might be needed to update the table entirely.

- **OWNER TO new_owner**

  Changes the owner of a table, sequence, or view to the specified user.

- **SET TABLESPACE new_tablespace**

  Changes the table's tablespace to the specified tablespace and moves the data files associated with the table to the new tablespace. Indexes on the table, if any, are not moved; but they can be moved separately with additional **SET TABLESPACE** option in ALTER INDEX.

- **TO { GROUP groupname | NODE (nodename [, ...]) }**

  The syntax is only available in extended mode (when GUC parameter **support_extended_features** is **on**). Exercise caution when enabling the mode. It is mainly used for tools like internal dilatation tools. Common users should not use the mode.

- **ADD NODE (nodename [, ...])**

  It is only available for internal scale-out tools. Common users should not use the syntax.

- **DELETE NODE (nodename [, ...])**

  It is only available for internal scale-in tools. Common users should not use the syntax.

- **DISABLE TRIGGER [trigger_name | ALL | USER]**

  Disables a single trigger specified by **trigger_name**, disables all triggers, or disables only user triggers (excluding internally generated constraint triggers, for example, deferrable unique constraint triggers and exclusion constraints triggers).

  Exercise caution when using this function because data integrity cannot be ensured as expected if the triggers are not executed.

- **| ENABLE TRIGGER [trigger_name | ALL | USER]**

  Enables a single trigger specified by **trigger_name**, enables all triggers, or enables only user triggers.

- **| ENABLE REPLICA TRIGGER trigger_name**

  Determines that the trigger firing mechanism is affected by the configuration variable *session_replication_role*. When the replication role is **origin** (default value) or **local**, a simple trigger is fired.

  When **ENABLE REPLICA** is configured for a trigger, it is triggered only when the session is in replica mode.

- **| ENABLE ALWAYS TRIGGER trigger_name**

  Determines that all triggers are fired regardless of the current replica mode.

- **| { DISABLE | ENABLE } ROW LEVEL SECURITY**

  Enables or disables row-level security for a table.

  If row-level security is enabled for a data table but no row-level security policy is defined, the row-level access to the data table is not affected. If row-level security for a table is disabled, the row-level access to the table is not affected even if a row-level security policy has been defined. For details, see **CREATE ROW LEVEL SECURITY POLICY**.

- **| { NO FORCE | FORCE } ROW LEVEL SECURITY**

  Forcibly enables or disables row-level security for a table.

  By default, the table owner is not affected by the row-level security feature. However, if row-level security is forcibly enabled, the table owner (excluding system administrators) will be affected. System administrators are not affected by any row-level security policies.

- **| ENCRYPTION KEY ROTATION**

  Rotation of the transparent data encryption key.

  The data encryption key rotation of a table can be performed only when the TDE function is enabled for the database and **enable_tde** of the table is set to **on**. After the key rotation operation is performed, the system automatically applies for a new KMS key. After the key rotation, the data encrypted using the old key is decrypted using the old key, and the newly written data is encrypted using the new key. To ensure the security of encrypted data, you can periodically update the key based on the amount of new data in the encrypted table. It is recommended that the key be updated every two to three years.

- **REPLICA IDENTITY { DEFAULT | USING INDEX index_name | FULL | NOTHING }**

  Specifies the record level of old tuples in UPDATE and DELETE statements on a table in logical replication scenarios.

  - **DEFAULT** records the old value of the primary key column. If there is no primary key, **DEFAULT** does not record the old value.
  - **USING INDEX** records the old values of columns covered by the named indexes. These values must be unique, non-local, and non-deferrable, and contain the values of columns marked **NOT NULL**.
  - **FULL** records the old values of all columns in the row.

- **NOTHING** does not record information in old rows.

  In logical replication scenarios, when the UPDATE and DELETE statements of a table are parsed, the parsed old tuples consist of the information recorded in this method. For tables with primary keys, this option can be set to **DEFAULT** or **FULL**. For a table without a primary key, set this parameter to **FULL**. Otherwise, the old tuple will be parsed as empty during decoding. You are advised not to set this parameter to **NOTHING** in common scenarios because old tuples are always parsed as empty.

  For Ustore tables, the **NOTHING** option is invalid, and the actual effect is the same as that of **FULL**. If **DEFAULT** does not have a primary key, all columns in the row are recorded.

- **AUTO_INCREMENT [=] value**

  Sets the next auto-increment value of the auto-increment column. The configured value takes effect only when it is greater than the current auto-increment counter.

  The value must be a non-negative integer and cannot be greater than $2^{127} - 1$.

  This clause takes effect only when **sql_compatibility** is set to **B**.

- **[[DEFAULT] CHARACTER SET | CHARSET [=] default_charset] [[DEFAULT] COLLATE [=] default_collation]**

  Modifies the default character set and default collation of a table to the specified values. The modification does not affect the existing columns in the table.

- **CONVERT TO CHARACTER SET | CHARSET charset [COLLATE collation]**

  Modifies the default character set and default collation of a table to the specified values, sets the character set and collation of all columns with character type to the specified value and converts the data in the column to new character set encoding.

- **ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ON (EXPR)]**

  Adds an ILM policy to a table. A complete ILM policy consists of two parts: ILM action and ILM condition. The ILM action is used to define a specific data compression or movement behavior. The ILM condition is used to define a condition for triggering the ILM action. The ILM condition is a row-level condition, that is, when the ILM condition applies to each row in the heap table and the current row is not modified within a period of time, the ILM condition is met, and the ILM action is triggered. EXPR supports only basic operation functions (such as to_date and substr) of table columns and types.

- **[MODIFY { PARTITION | SUBPARTITION }] ILM { ENABLE | DISABLE | DELETE } POLICY policy_name**

  Modifies a single ILM policy of a table. **policy_name** is the value of **POLICY_NAME** queried from the system view GS_ADM_ILMOBJECTS or GS_MY_ILMOBJECTS.

- **[MODIFY { PARTITION | SUBPARTITION }] ILM { ENABLE_ALL | DISABLE_ALL | DELETE_ALL}**

  Modifies all ILM policies of a table.

– The column_clause can be one of the following clauses:

```
ADD [ COLUMN ] [ IF NOT EXISTS ] column_name data_type [ CHARACTER SET | CHARSET charset ] [ compress_mode ] [ COLLATE collation ] [ column_constraint [ ... ] ] [ FIRST | AFTER column_name ]    | MODIFY column_name data_type
| MODIFY column_name [ CONSTRAINT constraint_name ] NOT NULL [ ENABLE ]
| MODIFY column_name [ CONSTRAINT constraint_name ] NULL
| MODIFY [ COLUMN ] column_name data_type [ CHARACTER SET | CHARSET charset ] [{[ COLLATE collation ] | [ column_constraint ]} [ ... ] ] [ FIRST | AFTER column_name ]
| CHANGE [ COLUMN ] column_name new_column_name data_type [ CHARACTER SET | CHARSET charset ] [{[ COLLATE collation ] | [ column_constraint ]} [ ... ] ] [ FIRST | AFTER column_name ]
| DROP [ COLUMN ] [ IF EXISTS ] column_name [ RESTRICT | CASCADE ]
| ALTER [ COLUMN ] column_name [ SET DATA ] TYPE data_type [ COLLATE collation ] [ USING expression ]
| ALTER [ COLUMN ] column_name { SET DEFAULT expression | DROP DEFAULT }
```



```
| ALTER [ COLUMN ] column_name { SET | DROP } NOT NULL
| ALTER [ COLUMN ] column_name SET STATISTICS [ PERCENT ] integer
| ADD STATISTICS (( column_1_name, column_2_name [, ...] ))
| DELETE STATISTICS (( column_1_name, column_2_name [, ...] ))
| ALTER [ COLUMN ] column_name SET ( {attribute_option = value} [, ... ] )
| ALTER [ COLUMN ] column_name RESET ( attribute_option [, ... ] )
| ALTER [ COLUMN ] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }
| MODIFY column_name DROP IDENTITY
| MODIFY column_name GENERATED [ ALWAYS | BY DEFAULT [ ON NULL ] ] AS IDENTITY
[ ( identity_options ) ]
| ALTER [ COLUMN ] column_name ADD GENERATED [ ALWAYS | BY DEFAULT [ON NULL] ] AS
IDENTITY [ [ SEQUENCE NAME sequence_name] | ( identity_options ) ]
```


NOTE

- **ADD [COLUMN] [IF NOT EXISTS] column_name data_type [CHARACTER SET | CHARSET charset] [compress_mode] [COLLATE collation] [column_constraint [...]] [FIRST | AFTER column_name]**

  Adds a column to a table. If a column is added using ADD COLUMN, all existing rows in the table are initialized with the column's default value (**NULL** if no DEFAULT clause is specified). **FIRST | AFTER column_name** indicates that a column is added to a certain position. When IF NOT EXISTS is specified and columns with the same name exist, a notice is returned, indicating that the column already exists. When IF NOT EXISTS is not specified and columns with the same name exist, an error is returned.

- **ADD ({ column_name data_type [compress_mode] } [, ...])**

  Adds columns in the table.

- **MODIFY ({ column_name data_type | column_name [CONSTRAINT constraint_name] NOT NULL [ENABLE] | column_name [CONSTRAINT constraint_name] NULL } [, ...])**

  Modifies the data type of an existing column in the table. Running this command will clear the statistics of this column. You are advised to collect the statistics of this column again after the modification.

- **MODIFY [COLUMN] column_name data_type [CHARACTER SET | CHARSET charset] [{[COLLATE collation] | [column_constraint]} [...]] [FIRST | AFTER column_name]**

  Replaces the definition of existing field in a table with a new definition. The indexes and independent object constraints, such as primary keys, unique key and CHECK constraints in the previous field are not deleted. The [FIRST | AFTER column_name] syntax indicates that the position of a column in a table is changed when the column definition is modified.

  This syntax can be used only when **sql_compatibility** is set to **'B'**. Foreign tables are not supported. Encrypted columns cannot be modified. The data type and collation rule of partition key columns cannot be modified. The data type and collation rule of columns referenced by rules and materialized views cannot be modified.

  If a column whose data type or collation rule is modified is referenced by a generated column, the data in the generated column is regenerated.

  If some objects (such as indexes, independent object constraints, views, triggers, and row-level security policies) depend on a modified column, these objects are rebuilt during column modification. If the definition of the modified column violates the constraints of this type of object, the modification fails. For example, the data type of the column that is used as the view result column cannot be modified. Pay attention on the failure impact before modification.

  If a modified column is called by some objects (such as user-defined functions and stored procedures), the modified column does not process these objects. After the columns are modified, these objects may be unavailable. Please evaluate the impact before modification.

  Changing the character set or collation of a column converts the data in the column to the new character set for encoding.

  The syntax of this clause is the same as that of **MODIFY column_name data_type** in the previous clause, but the semantic function is different. When the GUC parameter **b_format_behavior_compat_options** contains the **enable_modify_column** option, the function of this clause is used.

  Running this command will clear the statistics of this column. You are advised to collect the statistics of this column again after the modification.

  This command refreshes statistics or historical statistics. More statistics are recorded, indicates that the cost of this command is higher. Therefore,

exercise caution when running this command in multi-partition and multi-historical statistics scenarios.

- **CHANGE [COLUMN] column_name new_column_name data_type [CHARACTER SET | CHARSET charset] [{[COLLATE collation] | [column_constraint]} [...]] [FIRST | AFTER column_name]**

  Replaces the definition and name of existing column in a table with a new definition and name. The new column name must be different from the previous. The indexes and independent object constraints, such as primary keys, unique key and CHECK constraints in the previous column are not deleted. The **[FIRST | AFTER column_name]** syntax indicates that the position of a column in a table is changed when the name and definition of the column are modified.

  This syntax can be used only when **sql_compatibility** is set to **'B'**. Foreign tables are not supported. Encrypted columns cannot be modified. The data type and collation rule of partition key columns cannot be modified. The data type and collation rule of columns referenced by rules and materialized views cannot be modified.

  If a column whose data type or collation rule is modified is referenced by a generated column, the data in the generated column is regenerated.

  If some objects (such as indexes, independent object constraints, views, triggers, and row-level security policies) depend on a modified column, these objects are rebuilt during column modification. If the definition of the modified column violates the constraints of this type of object, the modification fails. For example, the data type of the column that is used as the view result column cannot be modified. Pay attention on the failure impact before modification.

  If a modified column is called by some objects (such as user-defined functions and stored procedures), the modified column does not process these objects. After the column names are modified, these objects may be unavailable. Please evaluate the impact before modification.

  Modifying the character set or collation of a column converts the data in the column to the new character set for encoding.

- **DROP [COLUMN] [IF EXISTS] column_name [RESTRICT | CASCADE]**

  Drops a column from a table. Indexes and constraints related to the column are automatically dropped. If an object not belonging to the table depends on the column, **CASCADE** must be specified, such as a view.

  The DROP COLUMN statement does not physically remove the column, but simply makes it invisible to SQL operations. Subsequent INSERT and UPDATE operations on the table will store a **NULL** value for the column. Therefore, column deletion takes a short period of time but does not immediately release the tablespace on the disks, because the space occupied by the deleted column is not recycled. The space will be recycled when **VACUUM** is executed.

- **ALTER [COLUMN] column_name [SET DATA] TYPE data_type [COLLATE collation] [USING expression]**

  Modifies the type of a column in a table. Indexes and simple table constraints on the column will automatically use the new data type by reparsing the originally supplied expression.

  If the original data type of a column and the modified data type are binary compatible, you do not need to rewrite the entire table when running this statement. In other scenarios, the entire table is rewritten. You can check whether the original type and target type are binary compatible in the PG_CAST system catalog. If **castmethod** is **b**, they are binary compatible. For example, if the data type of the source table is text and is converted to int, table rewriting is triggered. If it is converted to clob, table rewriting is not triggered. If table rewriting is triggered, the deleted space on the table is recycled immediately.

Running this command will clear the statistics of this column. You are advised to collect the statistics of this column again after the modification.

- **ALTER [COLUMN] column_name { SET DEFAULT expression | DROP DEFAULT }**

  Sets or removes the default value for a column. The default values only apply to subsequent INSERT operations; they do not cause rows already in the table to change. Defaults can also be created for views, in which case they are inserted into INSERT statements on the view before the view's ON INSERT rule is applied.

- **ALTER [COLUMN] column_name { SET | DROP } NOT NULL**

  Changes whether a column is marked to allow null values or to reject null values. You can only use SET NOT NULL when the column contains no null values.

- **ALTER [COLUMN] column_name SET STATISTICS [PERCENT] integer**

  Specifies the per-column statistics-collection target for subsequent ANALYZE operations. The target can be set in the range from 0 to 10000. Set it to **–1** to revert to using the default system statistics target.

- **{ ADD | DELETE } STATISTICS ((column_1_name, column_2_name [, ...]))**

  Adds or deletes the declaration of collecting multi-column statistics to collect multi-column statistics as needed when ANALYZE is performed for a table or a database. If the GUC parameter **enable_functional_dependency** is disabled, the statistics about a maximum of 32 columns can be collected at a time. If the GUC parameter **enable_functional_dependency** is enabled, the statistics about a maximum of 4 columns can be collected at a time. You are not allowed to add or delete such declaration for system catalogs or foreign tables.

- **ALTER [COLUMN] column_name SET ({attribute_option = value} [, ...])**

  ALTER [COLUMN] column_name RESET (attribute_option [, ...])

  Sets or resets per-attribute options.

  Currently, the only defined attribute options are **n_distinct** and **n_distinct_inherited**. **n_distinct** affects statistics of a table, while **n_distinct_inherited** affects the statistics of the table and its subtables. Currently, only **SET/RESET n_distinct** is supported, and **SET/RESET n_distinct_inherited** is forbidden.

- **ALTER [COLUMN] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }**

  Sets the storage mode for a column. It specifies whether this column is held inline or in an attached table, and whether the data should be compressed. Executing **SET STORAGE** does not change a table. It only specifies the recommended strategy for future table updates.

- **MODIFY column_name DROP IDENTITY**

  Cancels an **IDENTITY** column. This operation deletes only the identity attribute and related implicit sequence of the column.

- **MODIFY column_name GENERATED [ALWAYS | BY DEFAULT [ON NULL]] AS IDENTITY [(identity_options)]**

  Modifies the attributes and sequence parameters of the **IDENTITY** column. If the attributes of the **IDENTITY** column are not specified, the original attributes of the **IDENTITY** column are retained. The optional identity_options clause can be used to override sequence options.

- **ALTER [COLUMN] column_name ADD GENERATED [ALWAYS | BY DEFAULT [ON NULL]] AS IDENTITY [[SEQUENCE NAME sequence_name] | (identity_options)]**

  Sets an existing column to an **IDENTITY** column. The column must be of the smallint, integer, bigint, decimal, numeric, float, double precision, or real numeric type (decimal, numeric, float, double precision, or real numeric type

in A-compatible mode), and the default attribute is not set for the column. **sequence_name** specifies the name of the implicit sequence.

- **column_constraint** is as follows:

```
[ CONSTRAINT constraint_name ]
  { NOT NULL |
    NULL |
    CHECK ( expression ) |
    DEFAULT default_expr  |
    ON UPDATE update_expr |
    GENERATED ALWAYS AS ( generation_expr ) [STORED] |
    GENERATED [ ALWAYS | BY DEFAULT [ ON NULL ] ] AS IDENTITY
[ ( identity_options ) ] |
    AUTO_INCREMENT |
    COMMENT 'string' |
    UNIQUE [KEY] index_parameters |
    PRIMARY KEY index_parameters |
    ENCRYPTED WITH ( COLUMN_ENCRYPTION_KEY = column_encryption_key,
ENCRYPTION_TYPE = encryption_type_value ) |
    REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH
SIMPLE ]
      [ ON DELETE action ] [ ON UPDATE action ] }    [ DEFERRABLE | NOT DEFERRABLE
| INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

  - **identity_options** is as follows:

```
[ INCREMENT [ BY ] increment ] [ MINVALUE minvalue | NO MINVALUE |
NOMINVALUE] [ MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE] [ START
{ [ WITH ] start | WITH LIMIT VALUE } ] [ CACHE cache | NOCACHE ] [ [ NO ]
CYCLE | NOCYCLE] [ SCALE [ EXTEND | NO EXTEND] | NOSCALE ]
```

  - **update_expr** is as follows:

```
{ CURRENT_TIMESTAMP | LOCALTIMESTAMP | NOW() }
```

- **compress_mode** of a column is as follows:

```
{ DELTA | PREFIX | DICTIONARY | NUMSTR | NOCOMPRESS }
```

– **table_constraint_using_index** used to add the primary key constraint or unique constraint based on the unique index is as follows:

```
[ CONSTRAINT constraint_name ]
  { UNIQUE | PRIMARY KEY } USING INDEX index_name
  [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

– **table_constraint** is as follows:

```
[ CONSTRAINT [ constraint_name ] ]
  { CHECK ( expression ) |
    UNIQUE [ idx_name ][ USING method ] ( { { column_name [ ( length ) ] | ( expression ) }
[ ASC | DESC ] } [, ... ] ) index_parameters |
    PRIMARY KEY [ USING method ] ( { column_name [ ASC | DESC ] }[, ... ] ) index_parameters
|
    FOREIGN KEY [ idx_name ] ( column_name [, ... ] ) REFERENCES reftable [ ( refcolumn
[, ... ] ) ]
      [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ] [ ON UPDATE
action ] }
  [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
  { [ COMMENT 'string' ] [ ... ] }
```

index_parameters is as follows:

```
[ WITH ( {storage_parameter = value} [, ... ] ) ]
  [ USING INDEX TABLESPACE tablespace_name ]
```

- Rename a table. The renaming does not affect stored data.

```
ALTER TABLE [ IF EXISTS ] table_name
  RENAME [ TO | AS | = ]  new_table_name;
```


NOTE

If you specify this parameter in a version 5.7 B-compatible database (**sql_compatibility** set to **'B'**, **b_format_version** set to **'5.7'**, and **b_format_dev_version** set to **'s2'**), the following situations may occur:

- If the character string corresponding to the new table name starts with "#mysql50#" and is followed by other characters, "#mysql50#" will be ignored.
- If the old and new table names are the same, no error is reported.

- Rename the specified column in the table.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }
    RENAME [ COLUMN ] column_name TO new_column_name;
```

- Rename the constraint of the table.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }
    RENAME CONSTRAINT constraint_name TO new_constraint_name;
```

- Set the schema of the table.

```
ALTER TABLE [ IF EXISTS ] table_name
    SET SCHEMA new_schema;
```

NOTE

- The schema setting moves the table into another schema. Associated indexes and constraints owned by table columns are migrated as well. Currently, the schema for sequences cannot be changed. If the table has sequences, delete the sequences, and create them again or delete the ownership between the table and sequences. In this way, the table schema can be changed.
- To change the schema of a table, you must also have the CREATE permission on the new schema. To add the table as a new child of a parent table, you must own the parent table as well. To alter the owner, you must also be a direct or indirect member of the new owning role, and that role must have the CREATE permission on the table's schema. These restrictions enforce that the user can only rebuild and delete the table. However, a system administrator can alter the ownership of any table anyway.
- All the actions except for RENAME and SET SCHEMA can be combined into a list of multiple alterations to apply in parallel. For example, it is possible to add several columns or alter the type of several columns in a single command. This is useful with large tables, since only one pass over the tables need be made.
- Adding a CHECK or NOT NULL constraint will scan the table to validate that existing rows meet the constraint.
- Adding a column with a non-NULL default or changing the type of an existing column will rewrite the entire table. Rewriting a large table may take much time and temporarily needs doubled disk space.

- Add columns.

```
ALTER TABLE [ IF EXISTS ] table_name
    ADD ( { column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint
[ ... ] ]} [, ...] );
```

- Update columns.

```
ALTER TABLE [ IF EXISTS ] table_name
    MODIFY ( { column_name data_type | column_name [ CONSTRAINT constraint_name ] NOT NULL
[ ENABLE ] | column_name [ CONSTRAINT constraint_name ] NULL } [, ...] );
```

- Commit all DML transactions in the table when the SQL statement is executed.

```
ALTER TABLE [ IF EXISTS ] table_name GSIWAITALL;
```

NOTE

This syntax is internally called by the CREATE GLOBAL INDEX CONCURRENTLY function to synchronize lock wait during online creation of global secondary indexes. The centralized system does not support distributed global secondary indexes. Therefore, this syntax is not supported.

Parameters

- **IF EXISTS**

  Sends a notice instead of an error if no tables have identical names. The notice prompts that the table you are querying does not exist.

- **table_name [*] | ONLY table_name | ONLY (table_name)**

  table_name is the name of the table that you need to modify.

  If **ONLY** is specified, only the table is modified. If **ONLY** is not specified, the table and all subtables are modified. You can add the asterisk (*) option following the table name to specify that all subtables are scanned, which is the default operation.

- **constraint_name**
  - Specifies the name of an existing constraint to drop in the DROP CONSTRAINT operation.
  - Specifies the name of a new constraint in the ADD CONSTRAINT operation.

  NOTICE

  For a new constraint, **constraint_name** is optional in B-compatible mode (that is, **sql_compatibility** set to **'B'**). For other modes, **constraint_name** is required.

- **index_name**

  Specifies the index name.

  NOTICE

  In the ADD CONSTRAINT operation:
  - The **index_name** is supported only in B-compatible databases (that is, **sql_compatibility** set to **'B'**).
  - For foreign key constraints, if **constraint_name** and **index_name** are specified at the same time, **constraint_name** is used as the index name.
  - For a unique key constraint, if both **constraint_name** and **index_name** are specified, **index_name** is used as the index name.

- **USING method**

  Specifies the name of the index method to be used.

  For details about the value range, see USING method in **Parameters**.

NOTICE

In the ADD CONSTRAINT operation:

- The USING method is supported only in B-compatible databases (that is, **sql_compatibility** set to **'B'**).
- In B-compatible mode, if USING method is not specified, the default index method is B-tree for Astore or UB-tree for Ustore.
- If the storage mode of a table is Ustore and the constraint in the SQL statement is specified as USING BTREE, the underlying layer automatically creates the constraint as USING UBTREE.

- **ASC | DESC**

  ASC specifies an ascending (default) sort order. **DESC** specifies a descending sort order.

NOTICE

In ADD CONSTRAINT, ASC|DESC is supported only in B-compatible databases (**sql_compatibility** set to **'B'**).

- **expression**

  Specifies an expression index constraint based on one or more columns of the table. It must be written in parentheses.

NOTICE

Expression indexes in the UNIQUE constraint are supported only in B-compatible databases (that is, **sql_compatibility** set to **'B'**).

- **storage_parameter**

  Specifies the name of a storage parameter.

  TDE options:

  – **enable_tde** (Boolean type)

    Specifies whether to set a table as an encrypted table. This parameter applies only to row-store tables, segment-page tables, hash bucket tables, temporary tables, and unlogged tables. When setting **enable_tde** to **on**, ensure that the TDE function has been enabled using the GUC parameter **enable_tde** and the information for accessing the key service has been set using the GUC parameter **tde_key_info**. For details about how to use this parameter, see section "Transparent Data Encryption" in *Feature Guide*.

    Value range: **on** and **off**.

    on: TDE is enabled.

    off: TDE is disabled.

NOTE

- After the value is changed from **on** to **off**, the inserted or updated data is still encrypted when being written to the old page, and is not encrypted when being written to the new page generated after the switchover.
- After the value is changed from **off** to **on**, the inserted or updated data is not encrypted when being written to the old page, and is automatically encrypted when being written to the new page generated after the switchover.

That is, the encryption status of the new data page is the same as that after the TDE status switchover, and the encryption status of the old data page is the same as that before the TDE status switchover. You are advised to manually perform VACUUM FULL on the table after the encryption switchover to ensure that the encryption status of all data pages is consistent.

Default value: **off**

– **encrypt_algo** (string type)

Specifies the encryption algorithm of the encrypted table.

Value range: a string. The value can be **AES_128_CTR** or **SM4_CTR**.

Default value: **AES_128_CTR** if **enable_tde** is set to **on**; otherwise, null.

The following option is added for creating an index:

– **parallel_workers** (int type)

Number of bgworker threads started when an index is created. For example, value **2** indicates that two bgworker threads are started to create indexes concurrently.

Value range: [0,32]. The value **0** indicates that concurrent index creation is disabled.

Default value: If this parameter is not set, the concurrent index creation function is disabled.

– **hasuids** (Boolean type)

Default value: **off**

If this parameter is set to **on**, a unique table-level ID is allocated to a tuple when the tuple is updated.

– **statistic_granularity**

Records the default **partition_mode** when the table analyzes statistics. For details about **partition_mode**, see **ANALYZE|ANALYSE**. This parameter is invalid for non-partitioned tables.

Value range: See the value range of **partition_mode**.

Default value: **AUTO**

- **new_owner**

Specifies the name of the new table owner.

- **new_tablespace**

Specifies the new name of the tablespace to which the table belongs.

- **column_name**, **column_1_name**, **column_2_name**

Specifies the name of a new or existing column.

- **data_type**

Specifies the type of a new column or a new type of an existing column.

- **compress_mode**

  Specifies whether to compress a table column. The clause specifies the compression algorithm preferentially used by the column. Row-store tables do not support compression.

- **charset**

  Specifies the character set of a table column. If this parameter is specified separately, the collation of the table column is set to the default collation of the specified character set.

  This syntax is supported only in B-compatible databases (that is, **sql_compatibility** set to **'B'**).

- **collation**

  Specifies the collation rule name of a column. The optional COLLATE clause specifies a collation for the new column; if omitted, the collation is the default for the new column. You can run the **select * from pg_collation;** command to query collation rules from the pg_collation system catalog. The default collation rule is the row starting with **default** in the query result.

  In a B-compatible database (that is, **sql_compatibility** set to **'B'**), the utf8mb4_bin, utf8mb4_general_ci, utf8mb4_unicode_ci, binary, gbk_chinese_ci, gbk_bin, gb18030_chinese_ci, and gb18030_bin collations are also supported. For details, see **Table 7-220**.

  NOTE

  - Only the character type supports the specified character set. If the binary character set or collation is specified, the character type is converted to the corresponding binary type. If the type mapping does not exist, an error is reported. Currently, only the mapping from the TEXT type to the BLOB type is available.
  - Except the binary character set and collation, only the character set that is the same as the database encoding can be specified.
  - If the character set or collation of a column is not explicitly specified and the default character set or collation of the table is specified, the character set or collation of the column is inherited from the table. If the default character set or collation of a table does not exist, the character set and collation of table columns inherit the character set and collation of the current database when **b_format_behavior_compat_options** contains **'default_collation'**.
  - If the character set corresponding to the modified character set or collation is different from the character set of current column, the data in the column is converted to the specified character set for encoding.

- **USING expression**

  Specifies how to compute the new column value from the old; if omitted, the default conversion is an assignment cast from old data type to new. A USING clause must be provided if there is no implicit or assignment cast from the old to new type.

NOTE

USING in ALTER TYPE can specify any expression involving the old values of the row; that is, it can refer to any columns other than the one being cast. This allows general casting to be done with the ALTER TYPE syntax. Because of this flexibility, the USING expression is not applied to the column's default value (if any); the result might not be a constant expression as required for a default. This means that when there is no implicit or assignment cast from old to new type, ALTER TYPE might fail to convert the default even though a USING clause is supplied. In such cases, drop the default with DROP DEFAULT, perform ALTER TYPE, and then use SET DEFAULT to add a suitable new default. Similar considerations apply to indexes and constraints involving the column.

- **NOT NULL | NULL**

  Sets whether the column allows null values.

- **ENABLE**

  Specifies that the constraint is enabled. By default, the constraint is enabled.

- **integer**

  Specifies the constant value of a signed integer. When using **PERCENT**, the range of **integer** is from 0 to 100.

- **attribute_option**

  Specifies an attribute option.

- **PLAIN | EXTERNAL | EXTENDED | MAIN**

  Specifies a column-store mode.

  – **PLAIN** must be used for fixed-length values (such as integers). It must be inline and uncompressed.

  – **MAIN** is for inline, compressible data.

  – **EXTERNAL** is for external, uncompressed data. Use of **EXTERNAL** will make substring operations on **text** and **bytea** values run faster, at the penalty of increased storage space.

  – **EXTENDED** is for external, compressed data. **EXTENDED** is the default for most data types that support non-**PLAIN** storage.

- **CHECK (expression)**

  New rows or rows to be updated must satisfy for an expression to be true. If any row produces a false result, an error is raised and the database is not modified.

  A check constraint specified as a column constraint should reference only the column's values, while an expression in a table constraint can reference multiple columns.

  Currently, **CHECK (expression)** does not include subqueries and cannot use variables apart from the current column.

- **DEFAULT default_expr**

  Assigns a default data value to a column.

  The data type of the default expression must match the data type of the column.

  The default expression will be used in any insert operation that does not specify a value for the column. If there is no default value for a column, then the default value is **NULL**.

- **ON UPDATE update_expr**

  The ON UPDATE clause is an attribute constraint of a column.

  When an UPDATE operation is performed on a tuple in a table, if new values of updated columns are different from old values in the table, column values with this attribute but not in updated columns are automatically updated to the current timestamp. If new values of updated columns are the same as old values in the table, column values with this attribute but not in updated columns remain unchanged. If columns with this attribute are in updated columns, column values are updated according to the specified update value.

  NOTE

  - This attribute can be specified only in B-compatible database 5.7 (that is, **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**).
  - In terms of syntax, **update_expr** supports three keywords: CURRENT_TIMESTAMP, LOCALTIMESTAMP, and NOW(). You can also specify or not specify the precision of a keyword with parentheses. For example, ON UPDATE CURRENT_TIMESTAMP(), ON UPDATE CURRENT_TIMESTAMP(5), ON UPDATE LOCALTIMESTAMP(), and ON UPDATE LOCALTIMESTAMP(6). If the keyword does not contain parentheses or contains empty parentheses, the precision is 0. The NOW keyword cannot contain parentheses. The three types of keywords are synonyms of each other and have the same attribute effect.
  - This attribute can be specified only for columns of the following types: timestamp, datetime, date, time without time zone, smalldatetime, and abstime.
  - The CREATE TABLE AS syntax does not inherit the column attributes.
  - The CREATE TABLE LIKE syntax can use INCLUDING UPDATE or EXCLUDING UPDATE to inherit or exclude a constraint. The LIKE syntax is inherited from the LIKE syntax of PostgreSQL. Currently, the ILM policy information of the old table cannot be copied.
  - The precision specified by this attribute can be different from the precision specified by the type in the corresponding column. After the column value is updated through this attribute, the minimum precision is displayed. For example, ALTER TABLE t1 ADD col1 timestamp(6) ON UPDATE CURRENT_TIMESTAMP(3). If the UPDATE syntax triggers the attribute to take effect, the values of **col1** are displayed with three decimal places after the update.
  - The same column cannot be specified for this attribute and the generated column constraint at the same time.
  - This attribute cannot be specified for the partition key in a partitioned table.

- **COLUMN_ENCRYPTION_KEY = column_encryption_key**

  Specifies the name of the CEK in the ENCRYPTED WITH constraint.

  Value range: a string. It must comply with the **naming convention**.

- **ENCRYPTION_TYPE = encryption_type_value**

  For the encryption type in the ENCRYPTED WITH constraint, the value of **encryption_type_value** is **DETERMINISTIC** or **RANDOMIZED**.

- **GENERATED ALWAYS AS (generation_expr) [STORED]**

  This clause creates a column as a generated column. The value of the generated column is calculated by **generation_expr** when data is written (inserted or updated). **STORED** indicates that the value of the generated column is stored as a common column.

NOTE

- The STORED keyword can be omitted, which has the same semantics as not omitting STORED.
- The generation expression cannot refer to data other than the current row in any way. The generation expression cannot reference other generation columns or system columns. The generation expression cannot return a result set. No subquery, aggregate function, or window function can be used. The function called by the generation expression can only be an immutable function.
- Default values cannot be specified for generated columns.
- The generated column cannot be used as a part of the partition key.
- Do not specify the generated column and the CASCADE, SET NULL, and SET DEFAULT actions of the ON UPDATE constraint clause at the same time. Do not specify the generated column and the SET NULL, and SET DEFAULT actions of the ON DELETE constraint clause at the same time.
- The method of modifying and deleting generated columns is the same as that of common columns. If you delete a common column that a generated column depends on, the generated column is automatically deleted. The type of the column on which the generated column depends cannot be changed.
- The generated column cannot be directly written. In the INSERT or UPDATE statement, values cannot be specified for generated columns, but the keyword DEFAULT can be specified.
- The permission control for generated columns is the same as that for common columns.
- Columns cannot be generated for . In foreign tables, only **postgres_fdw** supports generated columns.

- **GENERATED [ALWAYS | BY DEFAULT [ON NULL]] AS IDENTITY [(identity_options)]**

  Creates a column as an **IDENTITY** column. An implicit sequence is automatically created based on identity_options and attached to a specified column. When data is inserted, the value obtained from the sequence is automatically assigned to the column.

  – **GENERATED [ALWAYS] AS IDENTITY**: Only identity values provided by the sequence generator can be inserted into this column. User-specified values cannot be inserted into this column.

  – **GENERATED BY DEFAULT AS IDENTITY**: User-specified values are preferentially inserted into this column. If no value is specified, the identity values provided by the sequence generator are inserted.

  – **GENERATED BY DEFAULT ON NULL AS IDENTITY**: User-specified values are preferentially inserted into this column. If a null value is specified or no value is specified, the identity values provided by the sequence generator are inserted.

  The optional identity_options clause can be used to override sequence options.

  – **increment**: specifies the implicit sequence step. If the value is a positive number, an ascending sequence is generated. If the value is a negative number, a descending sequence is generated. The default value is **1**.

  – **MINVALUE minvalue | NO MINVALUE | NOMINVALUE**: specifies the minimum value of an execution sequence. If **minvalue** is not specified or **NO MINVALUE** is specified, the default value of an ascending sequence is **1**, and the default value of a descending sequence is **$-10^{27} + 1$**. **NOMINVALUE** is equivalent to **NO MINVALUE**.

- **MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE**: specifies the maximum value of an execution sequence. If **maxvalue** is not specified or **NO MAXVALUE** is specified, the default value of an ascending sequence is $10^{28} - 1$, and the default value of a descending sequence is **–1**. **NOMAXVALUE** is equivalent to **NO MAXVALUE**.
- **[START { [WITH] start | WITH LIMIT VALUE }]**: **start** specifies the start value of the specified implicit sequence. Default value: **minvalue** for ascending sequences and **maxvalue** for descending sequences. If **START WITH LIMIT VALUE** is declared, the maximum (for ascending sequences) or minimum (for descending sequences) value in the table is specified as the start value of an implicit sequence.
- **cache**: specifies the number of sequences stored in the memory for quick access purposes. The default value is **1**, indicating that only one value can be generated at a time, that is, no cache is generated.
- **NOCACHE**: No value of the sequence is stored in advance.
- **CYCLE**: recycles sequences after the number of sequences reaches **maxvalue** or **minvalue**. If **NO CYCLE** is declared, any call to nextval returns an error after the sequence reaches its maximum or minimum value. **NOCYCLE** is equivalent to **NO CYCLE**. The default value is **NO CYCLE**.
- **SCALE**: enables sequence scalability. If specified, a numeric offset is appended to the beginning of the sequence to prevent duplicate items in the generated value. If **NOSCALE** is declared, sequence scalability is disabled. The default value is **NOSCALE**.
- **EXTEND**: extends the numeric offset length (default value: **6**), aligns the sequence generation value to **x** (default value: **6**) + **y** (maximum number of digits). **SCALE** must be specified when **EXTEND** is specified. If **NOEXTEND** is declared, the numeric offset length is not extended. The default value is **NOEXTEND**.

NOTE

- The **IDENTITY** column can only be of the smallint, integer, bigint, decimal, numeric, float, double precision, or real type.
- In A-compatible mode, when an **IDENTITY** column of the integer type is created, the **IDENTITY** column is of the numeric type by default.
- The method of changing an **IDENTITY** column is the same as that of changing a common column, but the type can be changed only to smallint, integer, bigint, decimal, numeric, float, double precision, or real.
- By default, the **IDENTITY** column has the NOT NULL constraint.
- A table can contain only one **IDENTITY** column.
- The method of deleting an **IDENTITY** column is the same as that of deleting a common column. When a column is deleted, the implicit sequence of the **IDENTITY** column is automatically deleted.
- The **IDENTITY** column cannot be specified together with the SET DEFAULT action.
- The type of the implicit sequence that is automatically created is LARGE SEQUENCE.
- You cannot use DROP LARGE SEQUENCE or ALTER LARGE SEQUENCE to modify the implicit sequence of an identity.
- After permission is granted to the table, data can be inserted properly. To modify the **IDENTITY** column, delete the **IDENTITY** attribute, or delete the **IDENTITY** column, you need to grant permission to the corresponding implicit sequence.
- The [SCALE [EXTEND | NOEXTED] | NOSCALE] clause is available only when an **IDENTITY** column is created in a centralized system in A-compatible mode.
- In a fully-encrypted database, the encrypted **IDENTITY** column cannot be specified during table creation.

- **AUTO_INCREMENT**

  Specifies an auto-increment column.

  For details, see **AUTO_INCREMENT**.

- **COMMENT [=] 'string'**
  - The COMMENT [=] 'string' clause is used to add comments to a table.
  - The COMMENT 'string' in column_constraint indicates that comments are added to a column.
  - The COMMENT 'string' in table_constraint indicates that comments are added to the indexes corresponding to the primary key and unique key.

  For details, see **COMMENT [=] 'string'**.

- **UNIQUE [KEY] index_parameters**

  UNIQUE KEY can be used only when **sql_compatibility** is set to **'B'**, which has the same semantics as UNIQUE.

  UNIQUE specifies that a group of one or more columns of a table can contain only unique values.

- **UNIQUE [index_name][USING method]({{ column_name [(length)] | (expression) } [ASC | DESC] }[, ...]) index_parameters**

  Specifies that a group of one or more columns of a table can contain only unique values.

  column_name (length) indicates the prefix key. For details, see **column_name (length)**.

  index_name indicates the index name.

NOTICE

- The index_name is supported only in B-compatible databases (that is, **sql_compatibility = 'B'**).
- For a unique key constraint, if both constraint_name and index_name are specified, index_name is used as the index name.

- **PRIMARY KEY index_parameters**

  PRIMARY KEY [USING method] ({ column_name [ASC | DESC] } [, ...]) index_parameters

  Specifies that a column or columns of a table can contain only unique (non-duplicate) and non-null values.

- **REFERENCES reftable [(refcolum)] [MATCH matchtype] [ON DELETE action] [ON UPDATE action] (column constraint)**

  FOREIGN KEY (column_name [, ...]) REFERENCES reftable [(refcolumn [, ...])] [MATCH matchtype] [ON DELETE action] [ON UPDATE action] (table constraint)

  The foreign key constraint requires that the group consisting of one or more columns in the new table should contain and match only the referenced column values in the referenced table. If **refcolum** is omitted, the primary key of **reftable** is used. The referenced column should be the only column or primary key in the referenced table. A foreign key constraint cannot be defined between a temporary table and a permanent table.

  There are three types of matching between a reference column and a referenced column:

  – **MATCH FULL**: A column with multiple foreign keys cannot be **NULL** unless all foreign key columns are **NULL**.

  – **MATCH SIMPLE** (default): Any unexpected foreign key column can be **NULL**.

  – **MATCH PARTIAL**: This option is not supported currently.

  In addition, when certain operations are performed on the data in the referenced table, the operations are performed on the corresponding columns in the new table. **ON DELETE**: specifies the operations to be executed after a referenced row in the referenced table is deleted. **ON UPDATE**: specifies the operation to be performed when the referenced column data in the referenced table is updated. Possible responses to the **ON DELETE** and **ON UPDATE** clauses are as follows:

  – **NO ACTION** (default): When a foreign key is deleted or updated, an error indicating that the foreign key constraint is violated is created. If the constraint is deferrable and there are still any referenced rows, this error will occur when the constraint is checked.

  – **RESTRICT**: When a foreign key is deleted or updated, an error indicating that the foreign key constraint is violated is created. It is the same as **NO ACTION** except that the constraint is not deferrable.

  – **CASCADE**: deletes any row that references the deleted row from the new table, or update the field value of the referenced row in the new table to the new value of the referenced column.

  – **SET NULL**: sets the referenced columns to **NULL**.

  - **SET DEFAULT**: sets the referenced columns to their default values.
- **DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE**

  Sets whether the constraint can be deferrable.
  - **DEFERRABLE**: deferrable to the end of the transaction and checked using **SET CONSTRAINTS**.
  - **NOT DEFERRABLE**: checks immediately after the execution of each command.
  - **INITIALLY IMMEDIATE**: checks immediately after the execution of each statement.
  - **INITIALLY DEFERRED**: checks when the transaction ends.
- **WITH ({storage_parameter = value} [, ...])**

  Specifies an optional storage parameter for a table or an index.
- **tablespace_name**

  Specifies the name of the tablespace where the index locates.
- **new_table_name**

  Specifies the new table name.
- **new_column_name**

  Specifies the new name of a specific column in a table.
- **new_constraint_name**

  Specifies the new name of a table constraint.
- **new_schema**

  Specifies the new schema name.
- **CASCADE**

  Automatically drops objects that depend on the dropped column or constraint (for example, views referencing the column).
- **RESTRICT**

  Refuses to drop the column if the column is referenced by other columns or constraints. **RESTRICT** is the default option. If **CASCADE** is not specified, the value is **RESTRICT**. An example of the statement is as follows:

  ```
  ALTER TABLE table_name [DROP [column] col_name [CASCADE | RESTRICT]];
  ```

  table_name indicates the table name, and **col_name** indicates the column name.
- **FIRST**

  Adds or changes the column to the first place.
- **AFTER column_name**

  Adds or changes the column after the column specified by **column_name**.

  NOTE

  - It is supported only in B-compatible databases (that is, **sql_compatibility = 'B'**).
  - **FIRST | AFTER column_name** cannot be used in the encrypted column.
  - The position of a column in a table that depends on rules cannot be changed (including the addition and change of the column position).
  - **FIRST | AFTER column_name** cannot be used in the foreign table.
  - Columns of the SET type cannot be changed to the specified position.

- **schema_name**

  Specifies the schema name of a table.

- **IF NOT EXISTS**

  When IF NOT EXISTS is specified and columns with the same name exist, a notice is returned, indicating that the column already exists. When IF NOT EXISTS is not specified and columns with the same name exist, an error is returned.

- **[DEFAULT] CHARACTER SET | CHARSET [=] default_charset**

  Changes the default character set of the table. If this parameter is specified separately, the default collation of the table is set to the default collation of the specified character set.

  This syntax is supported only when **sql_compatibility** is set to **'B'**.

- **[DEFAULT] COLLATE [=] default_collation**

  Changes the default collation of the table. If this parameter is specified separately, the default character set of the table is set to the character set corresponding to the specified collation.

  This syntax is supported only when **sql_compatibility** is set to **'B'**. For details about the collation, see **Table 7-220**.

  NOTE

  If the character set or collation of a table is not explicitly specified and the default character set or collation of the schema is specified, the character set or collation of the table is inherited from the schema. If the default character set or collation of a schema does not exist, the character set and collation of the table inherit those of the current database when **b_format_behavior_compat_options** contains **'default_collation'**.

Examples of Modifying a Table

- Rename a table.

```
gaussdb=# CREATE TABLE aa(c1 int, c2 int);
gaussdb=# ALTER TABLE IF EXISTS aa RENAME TO test_alt1;
```

- Modify the schema of a table.

```
-- Create the test_schema schema.
gaussdb=# CREATE SCHEMA test_schema;

-- Change the schema of the test_alt1 table to test_schema.
gaussdb=# ALTER TABLE test_alt1 SET SCHEMA test_schema;

-- Query table information.
gaussdb=# SELECT schemaname,tablename FROM pg_tables WHERE tablename = 'test_alt1';
 schemaname  | tablename
-------------+-----------
 test_schema | test_alt1
(1 row)
```

- Change the owner of a table.

```
-- Create user test_user.
gaussdb=# CREATE USER test_user PASSWORD '********';

-- Change the owner of the test_alt1 table to test_user.
gaussdb=# ALTER TABLE IF EXISTS test_schema.test_alt1 OWNER TO test_user;

-- Query.
gaussdb=# SELECT tablename, schemaname, tableowner FROM pg_tables WHERE tablename =
'test_alt1';
 tablename | schemaname  | tableowner
```



```
-----------+-------------+------------
 test_alt1 | test_schema | test_user
(1 row)
```

- Modify the tablespace of a table.

```
-- Create the tbs_data1 tablespace.
gaussdb=# CREATE TABLESPACE tbs_data1 RELATIVE LOCATION 'tablespace1/tbs_data1';

-- Change the tablespace of the test_alt1 table to tbs_data1.
gaussdb=# ALTER TABLE test_schema.test_alt1 SET TABLESPACE tbs_data1;

-- Query.
gaussdb=# SELECT tablename, tablespace FROM pg_tables WHERE tablename = 'test_alt1';
 tablename | tablespace
-----------+------------
 test_alt1 | tbs_data1
(1 row)

-- Delete.
gaussdb=# DROP TABLE test_schema.test_alt1;
gaussdb=# DROP TABLESPACE tbs_data1;
gaussdb=# DROP SCHEMA test_schema;
gaussdb=# DROP USER test_user;
```

Examples of Modifying a Column

- Change column names.

```
-- Create a table.
gaussdb=# CREATE TABLE test_alt2(c1 INT,c2 INT);

-- Change column names.
gaussdb=# ALTER TABLE test_alt2 RENAME c1 TO id;
gaussdb=# ALTER TABLE test_alt2 RENAME COLUMN c2 to areaid;

-- Query.
gaussdb=# \d test_alt2
   Table "public.test_alt2"
 Column |  Type   | Modifiers
--------+---------+-----------
 id     | integer |
 areaid | integer |
```

- Add columns.

```
-- Add a column to the test_alt2 table.
gaussdb=# ALTER TABLE IF EXISTS test_alt2 ADD COLUMN name VARCHAR(20);

-- Query.
gaussdb=# \d test_alt2
           Table "public.test_alt2"
  Column  |         Type          | Modifiers
----------+-----------------------+-----------
 id       | integer               |
 areaid   | integer               |
 name     | character varying(20) |
```

- Add the **AUTO_INCREMENT** column.

```
-- Create a table and add an auto-increment column.
gaussdb=# CREATE DATABASE test DBCOMPATIBILITY = 'B';
gaussdb=# \c test
test=# CREATE TABLE test_autoinc(col1 int);

-- Insert a data record.
test=# INSERT INTO test_autoinc(col1) VALUES(1);

-- Add a local auto-increment column, which starts from 1.
test=# ALTER TABLE test_autoinc ADD COLUMN col int primary key AUTO_INCREMENT;
test=# SELECT col,col1 FROM test_autoinc ORDER BY 2,1;
 col | col1
-----+------
   1 |    1
```



```
(1 row)

-- Set the next auto-increment value to 10.
test=# ALTER TABLE test_autoinc AUTO_INCREMENT = 10;

-- Enter NULL to trigger auto-increment. The auto-increment value is 10.
test=# INSERT INTO test_autoinc(col, col1) VALUES(NULL,2);

-- Enter 0 to trigger auto-increment. The auto-increment value is 11.
test=# INSERT INTO test_autoinc(col, col1) VALUES(0,3);

test=# SELECT col,col1 FROM test_autoinc ORDER BY 2,1;
 col | col1
-----+------
   1 |    1
  10 |    2
  11 |    3
(3 rows)

-- Delete.
test=# DROP TABLE test_autoinc;

-- Switch to the default database. Change the database name based on actual situation.
test=# \c postgres;
gaussdb=# DROP DATABASE test;
```

- Modify the data type of a column.

```
-- Change the type of the name column in the test_alt2 table.
gaussdb=# ALTER TABLE test_alt2 MODIFY name VARCHAR(50);

-- Query.
gaussdb=# \d test_alt2
          Table "public.test_alt2"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 id     | integer               |
 areaid | integer               |
 name   | character varying(50) |
-- Change the type of the name column in the test_alt2 table.
gaussdb=# ALTER TABLE test_alt2 ALTER COLUMN name TYPE VARCHAR(25);

-- Query.
gaussdb=# \d test_alt2
          Table "public.test_alt2"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 id     | integer               |
 areaid | integer               |
 name   | character varying(25) |
```

- Delete a column.

```
-- Delete the areaid column from test_alt2.
gaussdb=# ALTER TABLE test_alt2 DROP COLUMN areaid;

-- Query.
gaussdb=# \d test_alt2
          Table "public.test_alt2"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 id     | integer               |
 name   | character varying(25) |
```

- Modify the column-store mode.

```
-- View table details.
gaussdb=# \d+ test_alt2
                          Table "public.test_alt2"
 Column |         Type          | Modifiers | Storage  | Stats target | Description
--------+-----------------------+-----------+----------+--------------+-------------
 id     | integer               |           | plain    |              |
 name   | character varying(25) |           | extended |              |
Has OIDs: no
```



```
Options: orientation=row, compression=no, storage_type=USTORE, segment=off

-- Change the storage mode of the name column in the test_alt2 table.
gaussdb=# ALTER TABLE test_alt2 ALTER COLUMN name SET STORAGE PLAIN;

-- Query.
gaussdb=# \d+ test_alt2
                        Table "public.test_alt2"
 Column |         Type          | Modifiers | Storage | Stats target | Description
--------+-----------------------+-----------+---------+--------------+-------------
 id     | integer               |           | plain   |              |
 name   | character varying(25) |           | plain   |              |
Has OIDs: no
Options: orientation=row, compression=no, storage_type=USTORE, segment=off

-- Delete.
gaussdb=# DROP TABLE test_alt2;
```

- Move a column to a specified position.

```
-- Create a B-compatible database.
gaussdb=# CREATE DATABASE test DBCOMPATIBILITY 'B';

-- Connect to the test database and create the tbl_test table.
gaussdb=# \c test
test=# CREATE TABLE tbl_test(id int, name varchar(20));

-- Change the type of the name column in the tbl_test table and move it to the beginning.
test=# ALTER TABLE tbl_test MODIFY COLUMN name varchar(25) FIRST;

-- Query.
test=# \d tbl_test;
          Table "public.tbl_test"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 name   | character varying(25) |
 id     | integer               |

-- Change the type of the name column in the tbl_test table and move it after the id column.
test=# ALTER TABLE tbl_test MODIFY COLUMN name varchar(10) AFTER id;

-- Query.
test=# \d tbl_test;
          Table "public.tbl_test"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 id     | integer               |
 name   | character varying(10) |

-- Delete the tbl_test table.
test=# DROP TABLE tbl_test;

-- Switch to the default database and delete the test database. (Switch to the corresponding
database as required.)
test=# \c postgres
gaussdb=# DROP DATABASE test;
```

Examples of Modifying a Constraint

- Add a not-null constraint to a column.

```
-- Create a table.
gaussdb=# CREATE TABLE test_alt3(pid INT, areaid CHAR(5), name VARCHAR(20));

Add a not-null constraint to pid.
gaussdb=# ALTER TABLE test_alt3 MODIFY pid NOT NULL;

-- Query.
gaussdb=# \d test_alt3
          Table "public.test_alt3"
 Column |         Type          | Modifiers
```



```
--------+-----------------------+-----------
 pid    | integer               | not null
 areaid | character(5)          |
 name   | character varying(20) |
```

- Cancel the not-null constraint on a column.

```
gaussdb=# ALTER TABLE test_alt3 MODIFY pid NULL;
-- Query.
gaussdb=# \d test_alt3
          Table "public.test_alt3"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 pid    | integer               |
 areaid | character(5)          |
 name   | character varying(20) |
```

- Modify the default value of a column.

```
-- Modify the default value of id in the test_alt3 table.
gaussdb=# ALTER TABLE test_alt3 ALTER COLUMN areaid SET DEFAULT '00000';

-- Query.
gaussdb=# \d test_alt3
                Table "public.test_alt3"
 Column |         Type          |        Modifiers
--------+-----------------------+-------------------------
 pid    | integer               |
 areaid | character(5)          | default '00000'::bpchar
 name   | character varying(20) |
-- Delete the default value of id.
gaussdb=# ALTER TABLE test_alt3 ALTER COLUMN areaid DROP DEFAULT;

-- Query.
gaussdb=# \d test_alt3
          Table "public.test_alt3"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 pid    | integer               |
 areaid | character(5)          |
 name   | character varying(20) |
```

- Add a table-level constraint.
  - Directly add a constraint.

    ```
    -- Add a primary key constraint to the table.
    gaussdb=# ALTER TABLE test_alt3 ADD CONSTRAINT pk_test3_pid PRIMARY KEY (pid);

    -- Query.
    gaussdb=# \d test_alt3
              Table "public.test_alt3"
     Column |         Type          | Modifiers
    --------+-----------------------+-----------
     pid    | integer               | not null
     areaid | character(5)          |
     name   | character varying(20) |
    Indexes:
        "pk_test3_pid" PRIMARY KEY, ubtree (pid) WITH (storage_type=USTORE) TABLESPACE
    pg_default
    ```

  - Create an index and then add constraints.

    ```
    -- Create a table.
    gaussdb=# CREATE TABLE test_alt4(c1 INT, c2 INT);

    -- Create an index.
    gaussdb=# CREATE UNIQUE INDEX pk_test4_c1 ON test_alt4(c1);

    -- Associate the created index when adding a constraint.
    gaussdb=# ALTER TABLE test_alt4 ADD CONSTRAINT pk_test4_c1 PRIMARY KEY USING INDEX
    pk_test4_c1;

    -- Query.
    gaussdb=# \d test_alt4
    ```



```
  Table "public.test_alt4"
 Column |  Type   | Modifiers
--------+---------+-----------
 c1     | integer | not null
 c2     | integer |
Indexes:
    "pk_test4_c1" PRIMARY KEY, ubtree (c1) WITH (storage_type=USTORE) TABLESPACE
pg_default

-- Delete.
gaussdb=# DROP TABLE test_alt4;
```

- Delete a table-level constraint.

```
-- Delete a constraint.
gaussdb=# ALTER TABLE test_alt3 DROP CONSTRAINT IF EXISTS pk_test3_pid;

-- Query.
gaussdb=# \d test_alt3
          Table "public.test_alt3"
 Column |          Type          | Modifiers
--------+------------------------+-----------
 pid    | integer                | not null
 areaid | character(5)           |
 name   | character varying(20) |

-- Delete.
gaussdb=# DROP TABLE test_alt3;
```

Helpful Links

CREATE TABLE and **DROP TABLE**

7.12.6.33 ALTER TABLE PARTITION

Description

Modifies table partitions, including adding, deleting, splitting, merging, clearing, exchanging, and renaming partitions, moving partition tablespaces, enabling and disabling the automatic partitioning function, and modifying partition attributes.

Precautions

- Only the partitioned table owner or a user granted with the ALTER permission can run the **ALTER TABLE PARTITION** command. When separation of duties is disabled, system administrators have this permission by default.
- The tablespace of the added partition cannot be PG_GLOBAL.
- The name of the added partition must be different from names of existing partitions in the partitioned table.
- The key value of the added partition must be consistent with the type of partition keys in the partitioned table.
- If a range partition is added, the key value of the added partition must be greater than the upper limit of the last range partition in the partitioned table.
- If a list partition is added, the key value of the added partition cannot be the same as that of an existing partition.
- Hash partitions cannot be added.
- If the number of partitions in the target partitioned table has reached the maximum (**1048575**), partitions cannot be added.

- If a partitioned table has only one partition, the partition cannot be deleted.
- Use PARTITION FOR() to choose partitions. The number of specified values in the brackets should be the same as the number of columns specified when you define a partition, and they must be consistent. ALTER TABLE processes the partition to which **partition_value** belongs.
- The **Value** partitioned table does not support the ALTER PARTITION operation.
- Partitions cannot be added to an interval partitioned table.
- Hash partitioned tables do not support splitting, combination, addition, or deletion of partitions.
- Deleting, splitting, merging, clearing, and exchanging partitions will invalidate global indexes. The UPDATE GLOBAL INDEX clause can be used to update the indexes synchronously.
- If the UPDATE GLOBAL INDEX clause is not used when you delete, split, merge, clear, or exchange partitions, concurrent DML services may report errors due to invalidated indexes.
- If **enable_gpi_auto_update** is set to **on**, the global index is automatically updated even if the UPDATE GLOBAL INDEX clause is not declared.
- If you add a row-level expression when adding or changing an ILM policy for a data object, note that the row-level expression supports only the functions listed in the whitelist. For details about the whitelist function list, see **Row Expression Function Whitelist**.
- To enable automatic range partitioning, ensure that no MAXVALUE partition key exists in the partitioned table.
- Only a single partition key is supported when automatic range partitioning is enabled.
- To enable automatic list partitioning, ensure that no partition whose partition key is DEFAULT exists in a partitioned table.

Syntax

Modifying partitions in a partitioned table includes the main syntax of modifying a table partition, and the syntaxes of modifying a table partition name, resetting the partition ID, and enabling or disabling the automatic partitioning function.

- Modify the syntax of the table partition.

```
ALTER TABLE [ IF EXISTS ] { table_name  [*] | ONLY table_name | ONLY ( table_name  )}
    action [, ... ];
```

action indicates the following clauses for maintaining partitions. For the partition continuity when multiple clauses are used for partition maintenance, GaussDB executes DROP PARTITION and ADD PARTITION first, and then the rest clauses in sequence.

```
  move_clause  |
  exchange_clause  |
  row_clause  |
  merge_clause  |
  modify_clause  |
  split_clause  |
  add_clause  |
  drop_clause  |
  truncate_clause  |
  ilm_clause  |
  set_partitioning_clause
```


- The move_clause syntax is used to move the partition to a new tablespace.

```
MOVE PARTITION { partion_name | FOR ( partition_value [, ...] ) } TABLESPACE tablespacename
```

- The exchange_clause syntax is used to move the data from an ordinary table to a specified partition.

```
EXCHANGE PARTITION { ( partition_name ) | partition_name | FOR ( partition_value [, ... ] ) }
   WITH TABLE {[ ONLY ] ordinary_table_name | ordinary_table_name * | ONLY
( ordinary_table_name )}
   [ { WITH | WITHOUT } VALIDATION ] [ VERBOSE ] [ UPDATE GLOBAL INDEX ] [ UPDATE
DISTRIBUTED GLOBAL INDEX | NO UPDATE DISTRIBUTED GLOBAL INDEX ]
```

  The ordinary table and partition whose data is to be exchanged must meet the following requirements:

  - The number of columns in the ordinary table is the same as that of the partition, and their information should be consistent, including: column name, data type, constraint, collation information, storage parameter, and compression information.
  - The compression information of the ordinary table and partition should be consistent.
  - The number of ordinary table indexes is the same as that of local indexes of the partition, and the index information is the same.
  - The number and information of constraints of the ordinary table and partition should be consistent.
  - An ordinary table cannot be a temporary table. A partitioned table can only be a range partitioned table, list partitioned table, hash partitioned table, or interval partitioned table.
  - When the built-in security policy is enabled, ordinary and partitioned tables cannot have dynamic data masking or row-level security constraints.

NOTICE

- When the exchange is done, the data and tablespace of the ordinary table and partition are exchanged. The statistics about ordinary tables and partitions become unreliable, and they should be analyzed again.
- A non-partition key cannot be used to create a local unique index. Therefore, if an ordinary table contains a unique index, data cannot be exchanged.

  To exchange data, you can create an intermediate table. You need to insert the partition data to the intermediate table, truncate the partition, insert the ordinary table data to the partitioned table, drop the ordinary table, and rename the intermediate table to complete data exchange.
- In a scenario where both an ordinary table and a partitioned table are Ustore tables, if the UB-tree index type (RCR or PCR, RCR by default) of the ordinary table is different from the local UB-tree index type (RCR or PCR, RCR by default) of the partitioned table, data cannot be exchanged.
- If the DROP COLUMN operation is performed on an ordinary or partitioned table, the deleted column still exists physically. Therefore, you need to ensure that the deleted column of the ordinary table is strictly aligned with that of the partition. Otherwise, the exchange will fail.
- The EXCHANGE PARTITION { (partition_name) | partition_name | FOR (partition_value [, ...]) } operation is available in B-compatible database (that is, **sql_compatibility** set to **'B'**). In other modes, only the EXCHANGE PARTITION { (partition_name) | FOR (partition_value [, ...]) } operation is available.
  - When **sql_compatibility** is set to **'B'**, if **partition_name** is the name of a level-1 partition, the level-1 partition and ordinary table are exchanged. If **partition_name** is the name of a level-2 partition, the level-2 partition and ordinary table are exchanged.
  - Level-1 partitions and ordinary tables cannot be exchanged in level-2 partitioned tables.

– The row_clause syntax is used to set row movement of a partitioned table.

```
{ ENABLE | DISABLE } ROW MOVEMENT
```

– The merge_clause syntax is used to merge partitions into one. The maximum number of source partitions that can be merged in a command is 300.

```
MERGE PARTITIONS { partition_name } [, ...] INTO PARTITION partition_name
   [ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ ON ( EXPR )] ] [ TABLESPACE tablespacename ] [ UPDATE GLOBAL INDEX ] [ UPDATE DISTRIBUTED GLOBAL INDEX | NO UPDATE DISTRIBUTED GLOBAL INDEX ]
```


NOTICE

- For range partitioning and interval partitioning, the ranges of the source partitions must increase continuously, and the partition name after MERGE can be the same as the name of the last source partition. For list partitioning, there is no such range requirement on the source partitions, and the partition name after MERGE can be the same as that of any source partition. If the partition name after MERGE is the same as that of a source partition, they are considered as the same partition.
- If the GUC parameter **enable_ilm** is not enabled and the merge_clause syntax is used to merge multiple partitions with the ILM policy into one partition, the new partition does not inherit the ILM policy.

CAUTION

Ustore tables do not support ALTER TABLE MERGE PARTITIONS in transaction blocks and stored procedures.

– The modify_clause syntax is used to set whether a partitioned index is usable.

```
MODIFY PARTITION partition_name { UNUSABLE LOCAL INDEXES | REBUILD UNUSABLE LOCAL INDEXES }
```

– The split_clause syntax is used to split one partition into multiple ones.

```
SPLIT PARTITION { partition_name | FOR ( partition_value [, ...] ) } { split_point_clause | no_split_point_clause } [ UPDATE GLOBAL INDEX ] [ UPDATE DISTRIBUTED GLOBAL INDEX | NO UPDATE DISTRIBUTED GLOBAL INDEX ]
```

NOTICE

- The partition name after SPLIT can be the same as the source partition name, but they are regarded as different partitions.
- If the GUC parameter **enable_ilm** is not enabled and the split_clause syntax is used to split a partition with the ILM policy into multiple partitions, the new partitions do not inherit the ILM policy.

▪ The split_point_clause syntax for specifying the split point for range partitioned tables and interval partitioned tables is as follows:

```
AT ( partition_value ) INTO ( PARTITION partition_name [ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ ON ( EXPR )] ] [ TABLESPACE tablespacename ] , PARTITION partition_name [ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ ON ( EXPR )] ] [ TABLESPACE tablespacename ] )
```


NOTICE

The size of the split point should be in the range of partition keys of the partition to be split. The split point can only split one partition into two new partitions.

- The no_split_point_clause syntax for not specifying the split point for range partitioned tables and interval partitioned tables is as follows:

```
INTO { ( partition_less_than_item [, ...] ) | ( partition_start_end_item [, ...] ) }
```

NOTICE

- The first new partition key specified by **partition_less_than_item** should be greater than that of the previously split partition (if any), and the last partition key specified by **partition_less_than_item** should equal that of the partition being split.
- The first new partition key specified by **partition_start_end_item** should equal that of the former partition (if any), and the last partition key specified by **partition_start_end_item** should equal that of the partition being split.
- **partition_less_than_item** supports a partition key with up to 16 columns, while **partition_start_end_item** supports a one-column partition key. For details about the supported data types, see **PARTITION BY RANGE [COLUMNS] (partition_key)**.
- **partition_less_than_item** and **partition_start_end_item** cannot be used in the same statement. There is no restriction on different SPLIT statements.

- The partition_less_than_item syntax is as follows (the range of the last partition is not defined, that is, the VALUES LESS THAN (partition_value) part is not defined; by default, the last partition inherits the upper boundary value of the range defined for the source partition):

```
PARTITION partition_name VALUES LESS THAN {( { partition_value | MAXVALUE }  [, ...] ) | MAXVALUE }
   [ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ ON ( EXPR )] ] [ TABLESPACE tablespacename ]
```

NOTICE

During range partitioning, the MAXVALUE keyword can be used without parentheses. It can be used only in B-compatible mode. It cannot be used in subpartitions of level-2 partitions or in scenarios where multi-column partition keys are used.

- The partition_start_end_item syntax is as follows. For details about the constraints, see **START END**.

```
PARTITION partition_name {
      {START(partition_value) END (partition_value) EVERY (interval_value)} |
```



```
    {START(partition_value) END ({partition_value | MAXVALUE})} |
    {START(partition_value)} |
    {END({partition_value | MAXVALUE})}
} [ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day |
month | year } OF { NO MODIFICATION } [ ON ( EXPR )] ] [TABLESPACE
tablespace_name]
```

- The split_point_clause syntax for specifying the split point for a list partitioned table is as follows:

```
VALUES ( partition_value_list ) INTO ( PARTITION partition_name [ ILM ADD POLICY
ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO
MODIFICATION } [ ON ( EXPR )]] [ TABLESPACE tablespacename ] , PARTITION
partition_name [ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW }
AFTER n { day | month | year } OF { NO MODIFICATION } [ ON ( EXPR )]] [ TABLESPACE
tablespacename ] )
```

NOTICE

The split point must be a non-empty true subset of the source partition. Specifying a split point can only split one partition into two partitions.

- The no_split_point_clause syntax for not specifying the split point for a list partitioned table is as follows:

```
INTO ( PARTITION partition_name VALUES (partition_value_list) [ ILM ADD POLICY ROW
STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO
MODIFICATION } [ ON ( EXPR )]] [ TABLESPACE tablespacename ][, ...] )
```

NOTICE

- The range of the last partition is not defined, that is, the VALUES (partition_value_list) part is not defined; the partition range is equal to the remaining set of the source partition excluding other level-2 partitions.
- If no split point is specified, each new partition must be a non-empty true subset of the source partition and does not overlap with each other.

– The add_clause syntax is used to add one or more partitions to a specified partitioned table.

```
ADD {{partition_less_than_item | partition_start_end_item| partition_list_item } |
    PARTITION({partition_less_than_item | partition_start_end_item | partition_list_item})}
```

NOTICE

- The PARTITION({partition_less_than_item | partition_start_end_item | partition_list_item}) syntax can be used only in B-compatible mode.
- The ALTER TABLE table_name ADD PARTITION (partition_definition1, partition_definition2,…); syntax cannot be used to add multiple partitions. Only the original syntax for adding multiple partitions is supported: ALTER TABLE table_name ADD PARTITION (partition_definition1), ADD PARTITION (partition_definition2), …

- The partition_list_item syntax is as follows:

```
PARTITION partition_name VALUES [IN] (list_values_clause)
    [ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ ON ( EXPR )] ] [ TABLESPACE tablespacename ]
```

NOTICE

- **partition_list_item** supports a maximum of 16 partition keys. For details about the supported data types, see **PARTITION BY LIST [COLUMNS] (partition_key)**.
- Interval and hash partitioned tables do not support partition addition.
- IN needs to be used in B-compatible mode and cannot be used in subpartitions of level-2 partitions.

- The drop_clause syntax is used to remove a partition from a specified partitioned table.

```
DROP PARTITION  { partition_name | FOR (  partition_value [, ...] )  } [ UPDATE GLOBAL INDEX ] [ UPDATE DISTRIBUTED GLOBAL INDEX | NO UPDATE DISTRIBUTED GLOBAL INDEX ]
```

NOTICE

- Hash partitioned table does not support partition deletion.
- If a partitioned table has only one partition, the partition cannot be deleted.

- The truncate_clause syntax is used to remove a specified partition from a partitioned table.

```
TRUNCATE PARTITION  { partition_name | FOR (  partition_value [, ...] )  } [ UPDATE GLOBAL INDEX ] [ UPDATE DISTRIBUTED GLOBAL INDEX | NO UPDATE DISTRIBUTED GLOBAL INDEX ]
```

- The ilm_clause syntax adds an ILM policy to a partition. It supports the syntax for the OLTP table compression feature of data lifecycle management.

```
MODIFY PARTITION partition_name ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ ON ( EXPR ) ]
```

- The set_partitioning_clause syntax enables or disables automatic list/range partitioning.

```
SET PARTITIONING { AUTOMATIC | MANUAL }  |
SET INTERVAL ( [ interval_expr ] )
```

- The syntax for modifying the name of a partition is as follows:

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name  )}
    RENAME PARTITION { partion_name | FOR ( partition_value [, ...] ) } TO partition_new_name;
```

- Reset a partition ID.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name  )} RESET PARTITION;
```

Parameters

- **table_name**

  Specifies the name of a partitioned table.

  Value range: an existing table name

- **partition_name**

  Specifies the name of a partition.

Value range: an existing partition name

- **tablespacename**

  Specifies which tablespace the partition moves to.

  Value range: an existing tablespace name

- **partition_value**

  Specifies the key value of a partition.

  The value specified by PARTITION FOR (partition_value [, ...]) can uniquely identify a partition.

  Value range: partition keys for the partition to be operated.

- **UNUSABLE LOCAL INDEXES**

  Sets all the indexes unusable in the partition.

- **REBUILD UNUSABLE LOCAL INDEXES**

  Rebuilds all the indexes in the partition.

- **{ ENABLE | DISABLE } ROW MOVEMET**

  Sets row movement.

  If the tuple value is updated on the partition key during the UPDATE operation, the partition where the tuple is located is altered. Setting this parameter enables error messages to be reported or movement of the tuple between partitions.

  Value range:

  – **ENABLE**: Row movement is enabled.

  – **DISABLE**: Row movement is disabled.

  By default, this function is enabled.

- **ordinary_table_name**

  Specifies the name of the ordinary table whose data is to be migrated.

  Value range: an existing ordinary table name.

- **{ WITH | WITHOUT } VALIDATION**

  Checks whether the ordinary table data meets the specified partition key range of the partition to be migrated.

  Value range:

  – **WITH**: checks whether the ordinary table data meets the partition key range of the partition to be migrated. If any data does not meet the required range, an error is reported.

  – **WITHOUT**: does not check whether the ordinary table data meets the partition key range of the partition to be migrated.

  The default value is **WITH**.

  The check is time consuming, especially when the data volume is large. Therefore, use **WITHOUT** when you are sure that the current ordinary table data meets the partition key range of the partition to be migrated.

- **VERBOSE**

  When **VALIDATION** is **WITH**, if the ordinary table contains data that is out of the partition key range, insert the data to the correct partition. If there is no correct partition where the data can be inserted to, an error is reported.

NOTICE

Only when **VALIDATION** is **WITH**, **VERBOSE** can be specified.

- **partition_new_name**

  Specifies the new name of a partition.

  Value range: a string. It must comply with the **naming convention**.

- **UPDATE GLOBAL INDEX**

  Updates all global indexes in the partitioned table to ensure that correct data can be queried using global indexes. If this parameter is not used, all global indexes in the partitioned table become invalid.

- **UPDATE DISTRIBUTED GLOBAL INDEX | NO UPDATE DISTRIBUTED GLOBAL INDEX**

  (Optional) Rebuilds or invalidates global secondary indexes of the base table.

  The centralized mode does not support this option when you delete, split, merge, clear, or exchange partitions.

- **SET PARTITIONING { AUTOMATIC | MANUAL }**

  Enables or disables the function of automatic list partitioning. If the keyword **AUTOMATIC** is specified, the automatic partitioning function is enabled. If the keyword **MANUAL** is used, the automatic partitioning function is disabled.

- **SET INTERVAL ([interval_expr])**

  Implements the conversion between the interval partition and the range partition. The interval partition is equivalent to the range partition after the automatic partitioning function is enabled. If **interval_expr** is set to the default value, an interval partition is converted to a range partition. Otherwise, a range partition is converted to an interval partition. In the preceding information, **interval_expr** indicates the interval for automatically creating partitions. The value must comply with the value type of **partition_key**. Currently, only the numeric type and date/time type are supported, for example, **1**, **'1 day'**, and **'1 month'**.

Examples

- Change the name of a table partition.

```
-- Create a pre-partitioned table.
gaussdb=# CREATE TABLE test_p1 (col1 INT, col2 INT) PARTITION BY RANGE (col1)
(
    PARTITION p1 VALUES LESS THAN (10),
    PARTITION p2 VALUES LESS THAN (20),
    PARTITION p3 VALUES LESS THAN (MAXVALUE)
);

-- Change the partition name.
gaussdb=# ALTER TABLE test_p1 RENAME PARTITION p3 TO pmax;

-- Query partition information.
gaussdb=# SELECT relname, boundaries, oid FROM pg_partition WHERE parentid='test_p1'::regclass
AND parttype <> 'r';
 relname | boundaries |  oid
---------+------------+-------
 p1      | {10}       | 17066
 p2      | {20}       | 17067
 pmax    | {NULL}     | 17068
(3 rows)
```


- Move partition tablespaces.

```
-- Create a partition.
gaussdb=# CREATE TABLESPACE tbs_data1 RELATIVE LOCATION 'tablespace1/tbs_data1';

-- Move partition tablespaces.
gaussdb=# ALTER TABLE test_p1 MOVE PARTITION P1 TABLESPACE tbs_data1;

-- Check the partition tablespace.
gaussdb=# SELECT relname, spcname FROM pg_partition t1, pg_tablespace t2 WHERE
T1.reltablespace=t2.oid and t1.parentid='test_p1'::regclass;
 relname |  spcname
---------+-----------
 p1      | tbs_data1
(1 row)
```

- Exchange partitions.

```
-- Create an ordinary table and insert data into the table.
gaussdb=# CREATE TABLE test_ep1(col1 INT,col2 INT);
gaussdb=# INSERT INTO test_ep1 VALUES (GENERATE_SERIES(1,30), 1000);

-- Migrate ordinary table data to a specified partition.
gaussdb=# ALTER TABLE test_p1 EXCHANGE PARTITION (p1) WITH TABLE test_ep1 VERBOSE;

-- Query.
gaussdb=# SELECT COUNT(*) FROM test_p1 PARTITION (p1);
 count
-------
     9
(1 row)

-- Delete the test_ep1 table.
gaussdb=# DROP TABLE test_ep1;
```

- Merge partitions.

```
-- Merge the p2 and pmax partitions in the test_p1 table into pmax.
gaussdb=# ALTER TABLE test_p1 MERGE PARTITIONS p2,pmax INTO PARTITION pmax;

-- View a partition.
gaussdb=# SELECT relname, boundaries, oid FROM pg_partition WHERE parentid='test_p1'::regclass
AND parttype <> 'r' order by 1;
 relname | boundaries |  oid
---------+------------+-------
 p1      | {10}       | 17066
 pmax    | {NULL}     | 17070
(2 rows)

-- Delete tables and tablespaces.
gaussdb=# DROP TABLE test_p1;
gaussdb=# DROP TABLESPACE tbs_data1;
```

- Split partitions.

```
-- Create a table.
gaussdb=# CREATE TABLE test_r1 (col1 INT,col2 INT) PARTITION BY RANGE (col1)(
    PARTITION p1   VALUES LESS THAN (10),
    PARTITION pmax VALUES LESS THAN (MAXVALUE)
);

-- Split partitions.
gaussdb=# ALTER TABLE test_r1 SPLIT PARTITION pmax AT (20) INTO (PARTITION p2, PARTITION
pmax);
gaussdb=# ALTER TABLE test_r1 SPLIT PARTITION pmax INTO (
    PARTITION p3   VALUES LESS THAN (30),
    PARTITION pmax VALUES LESS THAN (MAXVALUE)
);

-- Query.
gaussdb=# SELECT relname, boundaries, oid FROM pg_partition WHERE parentid='test_r1'::regclass
AND parttype <> 'r' order by 1;
 relname | boundaries |  oid
---------+------------+-------
```



```
 p1     | {10}       | 17088
 p2     | {20}       | 17090
 p3     | {30}       | 17092
 pmax   | {NULL}     | 17093
(4 rows)

-- Delete the test_r1 table.
gaussdb=# DROP TABLE test_r1;
-- Create a table.
gaussdb=# CREATE TABLE test_r2(col1 INT, col2 INT) PARTITION BY RANGE (col1)(
    PARTITION p1   START(1) END(10),
    PARTITION p2   START(10) END(20),
    PARTITION pmax START(20) END(MAXVALUE)
);

-- Split partitions.
gaussdb=# ALTER TABLE test_r2 SPLIT PARTITION pmax INTO (
    PARTITION p3 START(20) END(30),
    PARTITION pmax START(30) END (MAXVALUE)
);

-- Query.
gaussdb=# SELECT relname, boundaries, oid FROM pg_partition WHERE parentid='test_r2'::regclass
AND parttype <> 'r' order by 1;
 relname | boundaries |  oid
---------+------------+-------
 p1_0    | {1}        | 17112
 p1_1    | {10}       | 17113
 p2      | {20}       | 17114
 p3      | {30}       | 17116
 pmax    | {NULL}     | 17117
(5 rows)

-- Delete the test_r2 table.
gaussdb=# DROP TABLE test_r2;
-- Create a table.
gaussdb=# CREATE TABLE test_l1(col1 INT, col2 INT) PARTITION BY LIST(col1)(
    PARTITION p1   VALUES (10,20),
    PARTITION p2 VALUES (30,40)
);

-- Split partitions.
gaussdb=# ALTER TABLE test_l1 SPLIT PARTITION p1 VALUES (10) INTO (PARTITION p1_1, PARTITION
p1_2);
gaussdb=# ALTER TABLE test_l1 SPLIT PARTITION p2 INTO (PARTITION p3_1 VALUES(30), PARTITION
p3_2);

-- Query.
gaussdb=# SELECT relname, boundaries, oid FROM pg_partition WHERE parentid='test_l1'::regclass
AND parttype <> 'r' order by 1;
 relname | boundaries |  oid
---------+------------+-------
 p1_1    | {10}       | 17132
 p1_2    | {20}       | 17133
 p3_1    | {30}       | 17134
 p3_2    | {40}       | 17135
(4 rows)

-- Delete the test_l1 table.
gaussdb=# DROP TABLE test_l1;
```

- Add a partition.

```
-- Create a table.
gaussdb=# CREATE TABLE test_p2 (col1 INT, col2 INT) PARTITION BY RANGE (col1)(
    PARTITION p1 VALUES LESS THAN (10),
    PARTITION p2 VALUES LESS THAN (20)
);

-- Add a partition.
gaussdb=# ALTER TABLE test_p2 ADD PARTITION p3 VALUES LESS THAN (30);
```



```
-- Delete the test_p2 table.
gaussdb=# DROP TABLE test_p2;
-- Create a table.
gaussdb=# CREATE TABLE test_p3 (col1 INT, col2 INT) PARTITION BY LIST(col1)(
    PARTITION p1 VALUES (1),
    PARTITION p2 VALUES (2)
);

-- Add a partition.
gaussdb=# ALTER TABLE test_p3 ADD PARTITION p3 VALUES (3);

-- Delete the test_p3 table.
gaussdb=# DROP TABLE test_p3;
```

- Delete a partition.

```
-- Create a table.
gaussdb=# CREATE TABLE test_p4 (col1 INT, col2 INT) PARTITION BY LIST(col1)(PARTITION p1
VALUES (1),PARTITION p2 VALUES (2));

-- Delete the p2 partition from the test_p3 table.
gaussdb=# ALTER TABLE test_p4 DROP PARTITION p2;

-- Query.
gaussdb=# SELECT relname, boundaries, oid FROM pg_partition WHERE parentid='test_p4'::regclass;
 relname | boundaries |  oid
---------+------------+-------
 test_p4 |            | 17187
 p1      | {1}        | 17188
(2 rows)

-- Delete the test_p4 table.
gaussdb=# DROP TABLE test_p4;

-- Specify the partition value to delete a partition.
-- Create a table.
gaussdb=# CREATE TABLE test_p4 (col1 INT, col2 INT) PARTITION BY RANGE(col1)(PARTITION p1
VALUES LESS THAN(1),PARTITION p2 VALUES LESS THAN (2));

-- Delete the partition whose partition key is 1 from the test_p3 table.
gaussdb=# ALTER TABLE test_p4 DROP PARTITION FOR (1);

-- Query.
gaussdb=# SELECT relname, boundaries FROM pg_partition WHERE parentid='test_p4'::regclass order
by 1 desc;
 relname | boundaries
---------+------------
 test_p4 |
 p1      | {1}
(2 rows)

-- Delete the test_p4 table.
gaussdb=# DROP TABLE test_p4;
```

- Truncate a partition.

```
-- Create a table.
gaussdb=# CREATE TABLE test_p5 (col1 INT, col2 INT) PARTITION BY RANGE (col1)(
    PARTITION p1 VALUES LESS THAN (5),
    PARTITION p2 VALUES LESS THAN (10)
);

-- Insert data.
gaussdb=# INSERT INTO test_p5 VALUES (GENERATE_SERIES(1,9), 100);

-- View data in the p2 partition.
gaussdb=# SELECT * FROM test_p5 PARTITION (p2);
 col1 | col2
------+------
    5 |  100
    6 |  100
```



```
   7 |  100
   8 |  100
   9 |  100
(5 rows)

-- Clear the data in the p2 partition.
gaussdb=# ALTER TABLE test_p5 TRUNCATE PARTITION p2;

-- View data in the p2 partition.
gaussdb=# SELECT * FROM test_p5 PARTITION (p2);
 col1 | col2
------+------
(0 rows)

-- Delete the test_p5 table.
gaussdb=# DROP TABLE test_p5;
```

- Enable and disable the automatic list partitioning function.

```
-- Create a list partition.
gaussdb=# CREATE TABLE list_int (c1 int, c2 int)
PARTITION BY LIST (c1)
(
   PARTITION p1 VALUES (1, 2, 3),
   PARTITION p2 VALUES (4, 5, 6)
);

-- Enable the automatic list partitioning function.
gaussdb=# ALTER TABLE list_int SET PARTITIONING AUTOMATIC;

-- Disable the function of automatic list partitioning.
gaussdb=# ALTER TABLE list_int SET PARTITIONING MANUAL;

-- Deletes a list partition.
gaussdb=# DROP TABLE list_int;
```

- Enable and disable the automatic range partitioning function.

```
-- Create a range partition.
gaussdb=# CREATE TABLE range_int (c1 int, c2 int)
PARTITION BY RANGE (c1)
(
   PARTITION p1 VALUES LESS THAN (5),
   PARTITION p2 VALUES LESS THAN (10),
   PARTITION p3 VALUES LESS THAN (15)
);

-- Enable the automatic range partitioning function.
gaussdb=# ALTER TABLE range_int SET INTERVAL (5);

-- Disable the automatic range partitioning function.
gaussdb=# ALTER TABLE range_int SET INTERVAL ();

-- Delete a range partition.
gaussdb=# DROP TABLE range_int;
```

Helpful Links

CREATE TABLE PARTITION and **DROP TABLE**

7.12.6.34 ALTER TABLE SUBPARTITION

Description

Modifies partitions of a level-2 partitioned table, including adding, deleting, clearing, splitting, merging, exchanging, and renaming partitions, moving partition tablespaces, enabling and disabling the automatic partitioning function, and modifying partition attributes.

Precautions

- The tablespace of the added partition cannot be PG_GLOBAL.
- The name of the added partition must be different from the names of the existing level-1 and level-2 partitions in the partitioned table.
- The key value of the added partition must be consistent with the type of partition keys in the partitioned table.
- If a range partition is added, the key value of the added partition must be greater than the upper limit of the last range partition in the partitioned table. To add a partition to a table with the **MAXVALUE** partition, you are advised to use the SPLIT syntax.
- If a list partition is added, the key value of the added partition cannot be the same as that of an existing partition. To add a partition to a table with the **DEFAULT** partition, you are advised to use the SPLIT syntax.
- Hash partitions cannot be added. However, if the level-2 partition mode of an level-2 partitioned table is hash but the level-1 partition mode is not hash, you can add a level-1 partition and create the corresponding level-2 partition.
- If the number of partitions in the target partitioned table has reached the maximum (**1048575**), partitions cannot be added.
- If the partitioned table contains only one level-1 or level-2 partition, the partition cannot be deleted.
- Hash partitions cannot be deleted.
- PARTITION FOR() and SUBPARTITION FOR() can be used to select partitions. The number of specified values in the brackets must be the same as the number of columns defined in partition creation.
- Only level-2 partitions (leaf nodes) can be split. Only range and list partitioning policies can be used and hash partitioning policies are not supported.
- Only level-2 partitions (leaf nodes) can be merged, and the source partitions must belong to the same level-1 partition.
- Only the owner of a partitioned table or users granted with the ALTER permission on the partitioned table can run the **ALTER TABLE PARTITION** command. The system administrator has the permission to run the command by default.
- Deleting, splitting, clearing, and exchanging partitions will invalidate the global index. The UPDATE GLOBAL INDEX clause can be used to update the index synchronously.
- If the UPDATE GLOBAL INDEX clause is not used when you delete, split, clear, or exchange partitions, concurrent DML services may report errors due to invalidated indexes.
- If **enable_gpi_auto_update** is set to **on**, the global index is automatically updated even if the UPDATE GLOBAL INDEX clause is not declared.
- If you add a row-level expression when adding or changing an ILM policy for a data object, note that the row-level expression supports only the functions listed in the whitelist. For details about the whitelist function list, see **Row Expression Function Whitelist**.
- To enable automatic level-1 or level-2 list partitioning, ensure that no DEFAULT partition key exists in the corresponding partitions.

Syntax

Modifying partitions in a level-2 partitioned table includes modifying the main syntax of a table partition, modifying the syntax of a table partition name, resetting the partition ID, and enabling or disabling the syntax of the automatic partitioning function.

- Modify the syntax of the table partition.

```
ALTER TABLE [ IF EXISTS ] { table_name  [*] | ONLY table_name | ONLY ( table_name  )}
    action [, ... ];
```

action indicates the following clauses for maintaining partitions. For the partition continuity when multiple clauses are used for partition maintenance, GaussDB executes DROP PARTITION and ADD PARTITION first, and then the rest clauses in sequence.

```
    move_clause  |
    exchange_clause  |
    row_clause  |
    merge_clause  |
    modify_clause  |
    add_clause    |
    drop_clause   |
    split_clause  |
    truncate_clause  |
    ilm_clause    |
    set_partitioning_clause
```

  - The move_clause syntax is used to move the partition to a new tablespace.

```
MOVE SUBPARTITION { subpartion_name | FOR ( subpartition_value [, ...] ) } TABLESPACE
tablespacename
```

  - The exchange_clause syntax is used to move the data from a general table to a specified partition.

```
EXCHANGE SUBPARTITION { ( subpartition_name ) | FOR ( subpartition_value [, ...] ) }
    WITH TABLE {[ ONLY ] ordinary_table_name | ordinary_table_name * | ONLY
( ordinary_table_name )}
    [ { WITH | WITHOUT } VALIDATION ] [ VERBOSE ] [ UPDATE GLOBAL INDEX ]
```

    The ordinary table and partition whose data is to be exchanged must meet the following requirements:

    - The number of columns of the ordinary table is the same as that of the partition, and their information should be consistent, including: column name, data type, constraint, collation information, storage parameter, and compression information.
    - The compression information of the ordinary table and partitioned table should be consistent.
    - The number of ordinary table indexes is the same as that of local indexes of the partition, and the index information is the same.
    - The number and information of constraints of the ordinary table and partition should be consistent.
    - The ordinary table cannot be a temporary table, and the partitioned table should be a level-2 partitioned table.
    - Ordinary tables and partitioned tables do not support dynamic data masking and row-level security constraints.

NOTICE

- When the exchange is done, the data and tablespace of the ordinary table and partitioned table are exchanged. In this case, the statistics on the ordinary table and partition become unreliable, and they should be analyzed again.
- A non-partition key cannot be used to create a local unique index. Therefore, if an ordinary table contains a unique index, data cannot be exchanged.

  To exchange data, you can create an intermediate table, insert partition data into the intermediate table, truncate partitions, insert ordinary table data into the partitioned table, drop the ordinary table, and rename the intermediate table.
- If the DROP COLUMN operation is performed on an ordinary or partitioned table, the deleted column still exists physically. Therefore, you need to ensure that the deleted column of the ordinary table is strictly aligned with that of the partition. Otherwise, the exchange will fail.

– The row_clause syntax is used to set row movement of a partitioned table.

```
{ ENABLE | DISABLE } ROW MOVEMENT
```

– The merge_clause syntax is used to merge partitions into one. The maximum number of source partitions that can be merged in a command is 300.

```
MERGE SUBPARTITIONS { subpartition_name } [, ...] INTO SUBPARTITION partition_name
    [ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ ON ( EXPR )]] [ TABLESPACE tablespacename ] [ UPDATE GLOBAL INDEX ]
```

NOTICE

- For range partitioning, the ranges of the source partitions must increase continuously, and the partition name after MERGE can be the same as the name of the last source partition. For list partitioning, there is no such range requirement on the source partitions, and the partition name after MERGE can be the same as that of any source partition. If the partition name after MERGE is the same as that of a source partition, they are considered as the same partition.
- If the GUC parameter **enable_ilm** is not enabled and the merge_clause syntax is used to merge multiple partitions with the ILM policy into one partition, the new partition does not inherit the ILM policy.

CAUTION

Ustore tables do not support ALTER TABLE MERGE SUBPARTITIONS in transaction blocks and stored procedures.

- The modify_clause syntax is used to set whether a partitioned index is usable. The syntax can be used in level-1 partitions.

  ```
  MODIFY PARTITION partition_name { UNUSABLE LOCAL INDEXES | REBUILD UNUSABLE LOCAL INDEXES }
  ```

  It can also be used in level-2 partitions.

  ```
  MODIFY SUBPARTITION partition_name { UNUSABLE LOCAL INDEXES | REBUILD UNUSABLE LOCAL INDEXES }
  ```

- The add_clause syntax is used to add one or more partitions to a specified partitioned table. The syntax can be used in level-1 partitions.

  ```
  ADD {partition_less_than_item | partition_list_item } [ ( subpartition_definition_list ) ]
  ```

  It can also be used in level-2 partitions.

  ```
  MODIFY PARTITION partition_name ADD subpartition_definition
  ```

  partition_less_than_item defines a range partition. The syntax is as follows:

  ```
  PARTITION partition_name VALUES LESS THAN ( partition_value | MAXVALUE )[ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ ON ( EXPR )] ] [ TABLESPACE tablespacename ]
  ```

  partition_list_item defines a list partition. The syntax is as follows:

  ```
  PARTITION partition_name VALUES ( partition_value [, ...] | DEFAULT ) [ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ ON ( EXPR )] ] [ TABLESPACE tablespacename ]
  ```

  subpartition_definition_list contains the **subpartition_definition** object of one or more level-2 partitions. The syntax is as follows:

  ```
  SUBPARTITION subpartition_name [ VALUES LESS THAN ( partition_value | MAXVALUE ) | VALUES ( partition_value [, ...] | DEFAULT )] [ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ ON ( EXPR )] ]  [ TABLESPACE tablespace ]
  ```

  NOTICE

  If the level-1 partition is a hash partition, you cannot use ADD to add a level-1 partition. If the level-2 partition is a hash partition, you cannot use MODIFY to add a level-2 partition.

- The drop_clause syntax is used to remove a partition from a specified partitioned table. The syntax can be used in level-1 partitions.

  ```
  DROP PARTITION  { partition_name | FOR (  partition_value )  } [ UPDATE GLOBAL INDEX ]
  ```

  It can also be used in level-2 partitions.

  ```
  DROP SUBPARTITION  { subpartition_name | FOR (  partition_value, subpartition_value )  } [ UPDATE GLOBAL INDEX ]
  ```

  NOTICE

  - If the level-1 partition is a hash partition, the level-1 partition cannot be deleted. If the level-2 partition is a hash partition, the level-2 partition cannot be deleted.
  - At least one sub-partition must be retained.

- The split_clause syntax is used to split one partition into partitions.

  ```
  SPLIT SUBPARTITION { subpartition_name| FOR ( subpartition_value [, ...] ) } { split_point_clause | no_split_point_clause } [ UPDATE GLOBAL INDEX ]
  ```


NOTICE

- The partition name after SPLIT can be the same as the source partition name, but they are regarded as different partitions.
- If the GUC parameter **enable_ilm** is not enabled and the split_clause syntax is used to split a partition with the ILM policy into multiple partitions, the new partitions do not inherit the ILM policy.

- The syntax for specifying the split point for range partitioning is as follows:

```
AT ( subpartition_value ) INTO ( SUBPARTITION subpartition_name [ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ ON ( EXPR )] ] [ TABLESPACE tablespacename ] , SUBPARTITION subpartition_name [ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ ON ( EXPR )] ] [ TABLESPACE tablespacename ] )
```

  NOTICE

  The split point must be in the range of partition keys of the partition to be split. Specifying a split point can only split one partition into two partitions.

- The syntax of not specifying the split point for range partitioning is as follows (the range of the last partition is not defined, that is, the VALUES LESS THAN (subpartition_value) part is not defined; by default, the last partition inherits the upper boundary value of the range defined for the source partition):

```
INTO ( SUBPARTITION subpartition_name VALUES LESS THAN (subpartition_value) [ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ ON ( EXPR )] ] [ TABLESPACE tablespacename ][, ...] )
```

  NOTICE

  - The range defined for the first new partition must be greater than that of the previous partition (if any) of the partition being split.
  - The range of the last new partition cannot be defined and it inherits the upper boundary value of the range defined for the source partition by default.
  - The new partition must meet the constraint of continuously increasing range.

- The syntax for specifying the split point for list-range partitioning is as follows:

```
VALUES ( subpartition_value ) INTO ( SUBPARTITION subpartition_name [ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ ON ( EXPR )] ] [ TABLESPACE tablespacename ] , SUBPARTITION subpartition_name [ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ ON ( EXPR )] ] [ TABLESPACE tablespacename ] )
```


NOTICE

The split point must be a non-empty true subset of the source partition. Specifying a split point can only split one partition into two partitions.

- The syntax for not specifying the split point for a list partitioned table is as follows:

```
INTO ( SUBPARTITION subpartition_name VALUES (subpartition_value_list) [ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ ON ( EXPR )] ] [ TABLESPACE tablespacename ][, ...] )
```

NOTICE

- The range of the last partition is not defined, that is, the VALUES (partition_value_list) part is not defined; the partition range is equal to the remaining set of the source partition excluding other level-2 partitions.
- If no split point is specified, each new partition must be a non-empty true subset of the source partition and does not overlap with each other.

– The truncate_clause syntax is used to remove a specified partition from a partitioned table. The syntax can be used in level-1 partitions.

```
TRUNCATE PARTITION  { partition_name | FOR (  partition_value [, ...] )  } [ UPDATE GLOBAL INDEX ]
```

It can also be used in level-2 partitions.

```
TRUNCATE SUBPARTITION  { subpartition_name | FOR (  subpartition_value [, ...] )  } [ UPDATE GLOBAL INDEX ]
```

– The ilm_clause syntax adds an ILM policy to a partition. It supports the syntax for the OLTP table compression feature of data lifecycle management. The syntax can be used in level-1 partitions.

```
MODIFY PARTITION partition_name ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ ON ( EXPR ) ]
```

It can also be used in level-2 partitions.

```
MODIFY SUBPARTITION subpartition_name ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ ON ( EXPR ) ]
```

– The set_partitioning_clause syntax enables or disables automatic list partitioning.

```
SET { PARTITIONING | SUBPARTITIONING } { AUTOMATIC | MANUAL }
```

- Modify the name of a partition. This syntax can be used to modify level-1 partitions of a partitioned table.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name  )}
    RENAME PARTITION { partion_name | FOR ( partition_value [, ...] ) } TO partition_new_name;
```

It can also be used to modify level-2 partitions of a partitioned table.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name  )}
    RENAME SUBPARTITION { subpartion_name | FOR ( subpartition_value [, ...] ) } TO subpartition_new_name;
```

- Reset a partition ID.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name  )} RESET PARTITION;
```


Parameters

- **table_name**

  Specifies the name of a partitioned table.

  Value range: an existing partitioned table name.

- **subpartition_name**

  Specifies the name of a level-2 partition name.

  Value range: an existing level-2 partition name.

- **tablespacename**

  Specifies which tablespace the partition moves to.

  Value range: an existing tablespace name.

- **partition_value**

  Specifies the key value of the level-1 partition.

  The value specified by PARTITION FOR (partition_value [, ...]) can uniquely identify a level-1 partition.

  Value range: partition keys for the level-1 partition to be operated.

- **subpartition_value**

  Specifies the level-1 and level-2 partition key values.

  The value specified by SUBPARTITION FOR (subpartition_value [, ...]) can uniquely identify a level-2 partition.

  Value range: partition key values of the level-1 and level-2 partitions for the level-2 partition to be operated.

- **UNUSABLE LOCAL INDEXES**

  Sets all the indexes unusable in the partition.

- **REBUILD UNUSABLE LOCAL INDEXES**

  Rebuilds all the indexes in the partition.

- **{ ENABLE | DISABLE } ROW MOVEMET**

  Specifies whether to enable row movement.

  If the tuple value is updated on the partition key during the UPDATE operation, the partition where the tuple is located is altered. Setting this parameter enables error messages to be reported or movement of the tuple between partitions.

  Value range:

  - **ENABLE**: Row movement is enabled.
  - **DISABLE**: Row movement is disabled.

  By default, this function is enabled.

- **ordinary_table_name**

  Specifies the name of the ordinary table whose data is to be migrated.

  Value range: an existing table name.

- **{ WITH | WITHOUT } VALIDATION**

  Checks whether the ordinary table data meets the specified partition key range of the partition to be migrated.

  Value range:

  - **WITH**: checks whether the ordinary table data meets the partition key range of the partition to be migrated. If any data does not meet the required range, an error is reported.
  - **WITHOUT**: does not check whether the ordinary table data meets the partition key range of the partition to be migrated.

  The default value is **WITH**.

  The check is time consuming, especially when the data volume is large. Therefore, use **WITHOUT** when you are sure that the current ordinary table data meets the partition key range of the partition to be migrated.

- **VERBOSE**

  When **VALIDATION** is **WITH**, if the ordinary table contains data that is out of the partition key range, insert the data to the correct partition. If there is no correct partition where the data can be inserted to, an error is reported.

  NOTICE

  Only when **VALIDATION** is **WITH**, **VERBOSE** can be specified.

- **partition_new_name**

  Specifies the new name of a partition.

  Value range: a string. It must comply with the **naming convention**.

- **subpartition_new_name**

  Specifies the new name of a level-2 partition.

  Value range: a string. It must comply with the **naming convention**.

- **UPDATE GLOBAL INDEX**

  If this parameter is used, all global indexes in the partitioned table are updated to ensure that data can be queried correctly using global indexes. If this parameter is not used, all global indexes in the partitioned table will become invalid.

- **SET { PARTITIONING | SUBPARTITIONING } { AUTOMATIC | MANUAL }**

  Enables or disables the function of automatic level-1/level-2 list partitioning. If the keyword **PARTITIONING** is used, the level-1 partition is specified. If the keyword **SUBPARTITIONING** is used, the level-2 partition is specified. If the keyword **AUTOMATIC** is used, the automatic partitioning function is enabled. If the keyword **MANUAL** is used, the automatic partitioning function is disabled.

Examples

- Rename a partition.

```
-- Create a level-2 partitioned table tbl_rge_lst_test.
gaussdb=# CREATE TABLE tbl_lst_reg_test(
    area_id char(5),
    sdate char(8),
    eid char(5),
    sales_amt int
) PARTITION BY LIST(area_id) SUBPARTITION BY RANGE(sdate)(
    PARTITION p_1001 VALUES ('1001')(
        SUBPARTITION p_1001_201901 VALUES LESS THAN ('20190201'),
        SUBPARTITION p_1001_201902 VALUES LESS THAN ('20190301'),
        SUBPARTITION p_1001_201903 VALUES LESS THAN ('20190401')
```



```
    ),
    PARTITION p_1002 VALUES ('1002')(
        SUBPARTITION p_1002_201901 VALUES LESS THAN ('20190201'),
        SUBPARTITION p_1002_201902 VALUES LESS THAN ('20190301'),
        SUBPARTITION p_100w VALUES LESS THAN ('20190401')
    )
);

-- Change the name of the subpartition p_100w to p_1002_201903.
gaussdb=# ALTER TABLE tbl_lst_reg_test RENAME SUBPARTITION p_100w TO p_1002_201903;

-- Query.
gaussdb=# SELECT table_name,partition_name,subpartition_name FROM db_tab_subpartitions
WHERE table_name = 'tbl_lst_reg_test' AND partition_name = 'p_1002';
    table_name    | partition_name | subpartition_name
------------------+----------------+-------------------
 tbl_lst_reg_test | p_1002         | p_1002_201901
 tbl_lst_reg_test | p_1002         | p_1002_201902
 tbl_lst_reg_test | p_1002         | p_1002_201903
(3 rows)
```

- Move partition tablespaces.

```
-- Create the tbs_data1 tablespace.
gaussdb=# CREATE TABLESPACE tbs_data1 RELATIVE LOCATION 'tablespace/tbs_data1';

-- Move the level-2 partition p_1002_201901 to the tablespace tbs_data1.
gaussdb=# ALTER TABLE tbl_lst_reg_test MOVE SUBPARTITION p_1002_201901 TABLESPACE
tbs_data1;

-- Move the level-2 partition p_1002_201902 to the tablespace tbs_data1.
gaussdb=# ALTER TABLE tbl_lst_reg_test MOVE SUBPARTITION FOR ('1002','20190325') TABLESPACE
tbs_data1;

-- Query the tablespace of the modified level-2 partition.
gaussdb=# SELECT subpartition_name,tablespace_name FROM db_tab_subpartitions WHERE
subpartition_name IN ('p_1002_201901','p_1002_201903');
 subpartition_name | tablespace_name
-------------------+-----------------
 p_1002_201901     | tbs_data1
 p_1002_201903     | tbs_data1
(2 rows)
```

- Exchange partitions.

```
-- Create an ordinary table and insert data into the table.
gaussdb=# CREATE TABLE tbl_test(
    area_id char(5),
    sdate char(8),
    eid char(5),
    sales_amt int
);

gaussdb=# INSERT INTO tbl_test VALUES ('1002','20190326','00001',9000);
gaussdb=# INSERT INTO tbl_test VALUES ('1002','20190326','00002',7500);
gaussdb=# INSERT INTO tbl_test VALUES ('1002','20190326','00003',6000);

-- Insert data into the partitioned table.
gaussdb=# INSERT INTO tbl_lst_reg_test VALUES ('1002','20190301','00001',126);

-- Exchange partitions.
gaussdb=# ALTER TABLE tbl_lst_reg_test EXCHANGE SUBPARTITION (p_1002_201903) WITH TABLE
tbl_test;

-- Query data. Data in the table and partition is exchanged.
gaussdb=# SELECT * FROM tbl_lst_reg_test;
 area_id |  sdate   |  eid  | sales_amt
---------+----------+-------+-----------
 1002    | 20190326 | 00001 |      9000
 1002    | 20190326 | 00002 |      7500
 1002    | 20190326 | 00003 |      6000
(3 rows)
```



```
gaussdb=# SELECT * FROM tbl_test;
 area_id |  sdate   |  eid  | sales_amt
---------+----------+-------+-----------
 1002    | 20190301 | 00001 |       126
(1 row)

-- Check that the partitioned table tablespace is exchanged.
gaussdb=# SELECT subpartition_name,tablespace_name FROM db_tab_subpartitions WHERE
subpartition_name = 'p_1002_201903';
 subpartition_name |  tablespace_name
-------------------+--------------------
 p_1002_201903     | DEFAULT TABLESPACE
(1 row)
```

- Merge partitions.

```
-- Merge partitions.
gaussdb=# ALTER TABLE tbl_lst_reg_test MERGE SUBPARTITIONS
p_1002_201901,p_1002_201902,p_1002_201903 INTO SUBPARTITION p_1002_20191;

-- Query level-2 partition information.
gaussdb=# SELECT table_name,partition_name,subpartition_name FROM db_tab_subpartitions
WHERE table_name = 'tbl_lst_reg_test';
    table_name    | partition_name | subpartition_name
------------------+----------------+-------------------
 tbl_lst_reg_test | p_1001         | p_1001_201901
 tbl_lst_reg_test | p_1001         | p_1001_201902
 tbl_lst_reg_test | p_1001         | p_1001_201903
 tbl_lst_reg_test | p_1002         | p_1002_20191
(4 rows)
```

- Add a partition.

```
-- Add a level-1 partition.
gaussdb=# ALTER TABLE tbl_lst_reg_test ADD PARTITION p_1003 VALUES('1003') (SUBPARTITION
p_1003_201901 VALUES LESS THAN ('20190201'));

-- Add a level-2 partition to a specific level-1 partition.
gaussdb=# ALTER TABLE tbl_lst_reg_test MODIFY PARTITION p_1003 ADD SUBPARTITION
p_1003_201902 VALUES LESS THAN ('20190301');

-- Query partition information.
gaussdb=# SELECT table_name,partition_name,subpartition_name FROM db_tab_subpartitions
WHERE table_name = 'tbl_lst_reg_test';
    table_name    | partition_name | subpartition_name
------------------+----------------+-------------------
 tbl_lst_reg_test | p_1001         | p_1001_201901
 tbl_lst_reg_test | p_1001         | p_1001_201902
 tbl_lst_reg_test | p_1001         | p_1001_201903
 tbl_lst_reg_test | p_1002         | p_1002_20191
 tbl_lst_reg_test | p_1003         | p_1003_201901
 tbl_lst_reg_test | p_1003         | p_1003_201902
(6 rows)
```

- Delete a partition.

```
-- Delete the level-2 partition p_1003_201902.
gaussdb=# ALTER TABLE tbl_lst_reg_test DROP SUBPARTITION p_1003_201902;

-- Query.
gaussdb=# SELECT table_name,partition_name,subpartition_name FROM db_tab_subpartitions
WHERE table_name = 'tbl_lst_reg_test' AND partition_name = 'p_1003';
    table_name    | partition_name | subpartition_name
------------------+----------------+-------------------
 tbl_lst_reg_test | p_1003         | p_1003_201901
(1 row)

-- Delete the level-1 partition p_1003.
gaussdb=# ALTER TABLE tbl_lst_reg_test DROP PARTITION p_1003;

-- Query.
gaussdb=# SELECT table_name,partition_name,subpartition_name FROM db_tab_subpartitions
WHERE table_name = 'tbl_lst_reg_test';
```



```
   table_name    | partition_name | subpartition_name
-----------------+----------------+-------------------
 tbl_lst_reg_test | p_1001         | p_1001_201901
 tbl_lst_reg_test | p_1001         | p_1001_201902
 tbl_lst_reg_test | p_1001         | p_1001_201903
 tbl_lst_reg_test | p_1002         | p_1002_20191
(4 rows)
```

- Split partitions.

```
-- Specify the split point to split partitions.
gaussdb=# ALTER TABLE tbl_lst_reg_test  SPLIT SUBPARTITION p_1002_20191
    AT ('20190201') INTO (SUBPARTITION p_1002_201901,SUBPARTITION p_1002_20191) UPDATE
GLOBAL INDEX;

-- Do not specify the split point to split partitions.
gaussdb=# ALTER TABLE tbl_lst_reg_test SPLIT SUBPARTITION p_1002_20191 INTO (
    SUBPARTITION p_1002_201902 VALUES LESS THAN ('20190301'),
    SUBPARTITION p_1002_201903
) UPDATE GLOBAL INDEX;

-- Query partition information.
gaussdb=# SELECT table_name,partition_name,subpartition_name,high_value FROM
db_tab_subpartitions;
   table_name    | partition_name | subpartition_name | high_value
-----------------+----------------+-------------------+------------
 tbl_lst_reg_test | p_1001         | p_1001_201901     | 20190201
 tbl_lst_reg_test | p_1001         | p_1001_201902     | 20190301
 tbl_lst_reg_test | p_1001         | p_1001_201903     | 20190401
 tbl_lst_reg_test | p_1002         | p_1002_201901     | 20190201
 tbl_lst_reg_test | p_1002         | p_1002_201902     | 20190301
 tbl_lst_reg_test | p_1002         | p_1002_201903     | 20190401
(6 rows)
```

- Clear partition data.

```
-- Clear the level-1 partition.
gaussdb=# ALTER TABLE tbl_lst_reg_test TRUNCATE PARTITION p_1001 UPDATE GLOBAL INDEX;

-- Clear the level-2 partition.
gaussdb=# ALTER TABLE tbl_lst_reg_test TRUNCATE SUBPARTITION p_1002_201903 UPDATE GLOBAL
INDEX;

-- Drop the table.
gaussdb=# DROP TABLE tbl_lst_reg_test;
gaussdb=# DROP TABLE tbl_test;

-- Drop the tablespace.
gaussdb=# DROP TABLESPACE tbs_data1;
```

Helpful Links

CREATE TABLE SUBPARTITION

7.12.6.35 ALTER TABLESPACE

Description

Modifies the attributes of a tablespace.

Precautions

- Only the tablespace owner or a user granted with the ALTER permission can run the **ALTER TABLESPACE** command. By default, system administrators have the permission. To modify a tablespace owner, you must be the tablespace owner or system administrator and a member of the **new_owner** role.

- The ALTER TABLESPACE operation on a row-store table cannot be performed in a transaction block.
- To change the owner, you must also be a direct or indirect member of the new owning role.

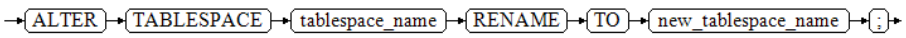
  NOTE

  If **new_owner** is the same as **old_owner**, the current user will not be verified. A message indicating successful ALTER execution is displayed.

Syntax

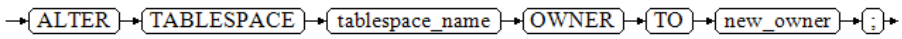
- The syntax of renaming a tablespace is as follows:

```
ALTER TABLESPACE tablespace_name
    RENAME TO new_tablespace_name;
```



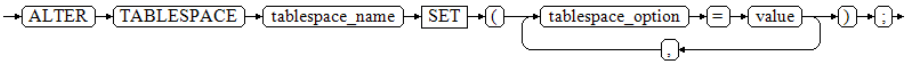
- The syntax of setting the owner of a tablespace is as follows:

```
ALTER TABLESPACE tablespace_name
    OWNER TO new_owner;
```



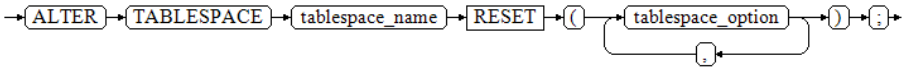
- The syntax of setting the attributes of a tablespace is as follows:

```
ALTER TABLESPACE tablespace_name
    SET ( { tablespace_option = value } [, ... ] );
```



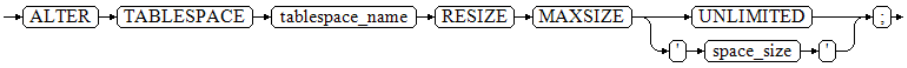
- The syntax of resetting the attributes of a tablespace is as follows:

```
ALTER TABLESPACE tablespace_name
    RESET ( { tablespace_option } [, ... ] );
```



- The syntax of setting the quota of a tablespace is as follows:

```
ALTER TABLESPACE tablespace_name
    RESIZE MAXSIZE { UNLIMITED | 'space_size'};
```



Parameters

- **tablespace_name**

  Specifies the tablespace to be modified.

  Value range: an existing table name

- **new_tablespace_name**

  Specifies the new name of the tablespace. The new name cannot start with **PG_**.

  Value range: a string. It must comply with the **naming convention**.

- **new_owner**

  Specifies the new owner of the tablespace.

  Value range: an existing username

- **tablespace_option**

  Sets or resets the parameters of a tablespace.

  Value range:

  - seq_page_cost: sets the optimizer to calculate the cost of obtaining disk pages in sequence. The default value is **1.0**.
  - **random_page_cost**: sets the optimizer to calculate the cost of obtaining disk pages in a non-sequential manner. The default value is **4.0**.

    NOTE

    - The value of **random_page_cost** is relative to that of **seq_page_cost**. It is meaningless when the value is equal to or less than the value of **seq_page_cost**.
    - The prerequisite for the default value **4.0** is that the optimizer uses indexes to scan table data and the hit ratio of table data in the cache is about 90%.
    - If the size of the table data space is smaller than that of the physical memory, decrease the value to a proper level. On the contrary, if the hit ratio of table data in the cache is lower than 90%, increase the value.
    - If random-access memory like SSD is adopted, the value can be decreased to a certain degree to reflect the cost of true random scan.

  Value range: Positive number of the floating-point type.

- **RESIZE MAXSIZE**

  Resets the maximum size of tablespace.

  Value range:

  - **UNLIMITED**: No limit is set for the tablespace.
  - Determined by **space_size**. For details about the format, see **CREATE TABLESPACE**.

    NOTE

    - If the adjusted quota is smaller than the current tablespace usage, the adjustment is successful. You need to decrease the tablespace usage to a value less than the new quota before writing data to the tablespace.

Examples

- Rename a tablespace.

```
-- Create a tablespace.
gaussdb=# CREATE TABLESPACE tbs_data1 RELATIVE LOCATION 'tablespace1/tbs_data1';

-- Rename a tablespace.
gaussdb=# ALTER TABLESPACE tbs_data1 RENAME TO tbs_data2;

-- Query.
gaussdb=# \db tbs_data2
            List of tablespaces
   Name    | Owner |       Location
-----------+-------+-----------------------
 tbs_data2 | omm   | tablespace1/tbs_data1
```

- Set the tablespace owner.

```
-- Create a user.
gaussdb=# CREATE USER test PASSWORD '********';

-- Change the owner of the tablespace.
gaussdb=# ALTER TABLESPACE tbs_data2 OWNER TO test;

-- Query.
```



```
gaussdb=# \db tbs_data2
            List of tablespaces
   Name    | Owner |       Location
-----------+-------+----------------------
 tbs_data2 | test  | tablespace1/tbs_data1
(1 row)
```

- Set tablespace attributes.

```
-- Change the value of seq_page_cost.
gaussdb=# ALTER TABLESPACE tbs_data2 SET (seq_page_cost = 10);

-- Query.
gaussdb=# SELECT * FROM pg_tablespace WHERE spcname = 'tbs_data2';
  spcname  | spcowner | spcacl |     spcoptions     | spcmaxsize | relative
-----------+----------+--------+--------------------+------------+----------
 tbs_data2 |    16778 |        | {seq_page_cost=10} |            | t
(1 row)
```

- Reset tablespace attributes.

```
-- Reset the value of seq_page_cost to the default value.
gaussdb=# ALTER TABLESPACE tbs_data2 RESET (seq_page_cost);

-- Query.
gaussdb=# SELECT * FROM pg_tablespace WHERE spcname = 'tbs_data2';
  spcname  | spcowner | spcacl | spcoptions | spcmaxsize | relative
-----------+----------+--------+------------+------------+----------
 tbs_data2 |    16778 |        |            |            | t
(1 row)
```

- Set the tablespace limit.

```
-- Set the maximum size of a tablespace.
gaussdb=# ALTER TABLESPACE tbs_data2 RESIZE MAXSIZE '10G';

-- Query.
gaussdb=# SELECT * FROM pg_tablespace WHERE spcname = 'tbs_data2';
  spcname  | spcowner | spcacl | spcoptions | spcmaxsize | relative
-----------+----------+--------+------------+------------+----------
 tbs_data2 |    16778 |        |            | 10485760 K | t
(1 row)

-- Delete the tablespace.
gaussdb=# DROP TABLESPACE tbs_data2;

-- Delete the user.
gaussdb=# DROP USER test;
```

Helpful Links

CREATE TABLESPACE and **DROP TABLESPACE**

7.12.6.36 ALTER TRIGGER

Description

ALTER TRIGGER is used to change the name of a trigger.

NOTE

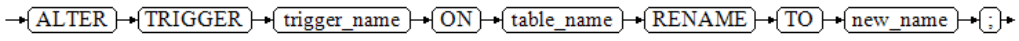
Currently, only the trigger name can be modified.

Precautions

The owner of the table where a trigger resides or a user granted the ALTER ANY SEQUENCE permission can perform the ALTER TRIGGER operation. A system administrator has this permission by default.

Syntax

```
ALTER TRIGGER trigger_name ON table_name RENAME TO new_name;
```



Parameters

- **trigger_name**

  Specifies the name of the trigger to be modified.

  Value range: an existing trigger

- **table_name**

  Specifies the name of the table where the trigger to be modified is located.

  Value range: an existing table having a trigger

- **new_name**

  Specifies the new name after modification.

  Value range: a string, which complies with the **naming convention**. A value contains a maximum of 63 characters and cannot be the same as other triggers on the same table.

Examples

```
-- Create a source table and a target table.
gaussdb=# CREATE TABLE test_trigger_src_tbl(id1 INT, id2 INT, id3 INT);
gaussdb=# CREATE TABLE test_trigger_des_tbl(id1 INT, id2 INT, id3 INT);

-- Create a DELETE trigger function.
gaussdb=# CREATE OR REPLACE FUNCTION TRI_DELETE_FUNC() RETURNS TRIGGER AS
        $$
        DECLARE
        BEGIN
                DELETE FROM test_trigger_des_tbl WHERE id1=OLD.id1;
                RETURN OLD;
        END
        $$ LANGUAGE plpgsql;

-- Create a DELETE trigger.
gaussdb=# CREATE TRIGGER delete_trigger BEFORE DELETE ON test_trigger_src_tbl FOR EACH ROW
EXECUTE PROCEDURE tri_delete_func();

-- Rename a trigger.
gaussdb=# ALTER TRIGGER delete_trigger ON test_trigger_src_tbl RENAME TO delete_trigger_renamed;

-- Delete the trigger.
gaussdb=# DROP TRIGGER delete_trigger_renamed ON test_trigger_src_tbl;

-- Delete the function.
gaussdb=# DROP FUNCTION tri_delete_func;

-- Delete the source table and target table.
gaussdb=# DROP TABLE test_trigger_src_tbl;
gaussdb=# DROP TABLE test_trigger_des_tbl;
```

Helpful Links

CREATE TRIGGER, **DROP TRIGGER**, and **ALTER TABLE**

7.12.6.37 ALTER TYPE

Description

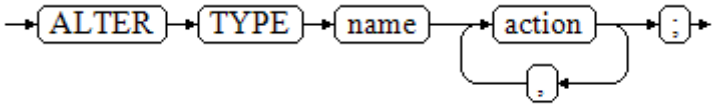
Modifies the definition of a type.

Precautions

Only the owner of a type, a user granted the ALTER permission on a type, or a user granted the ALTER ANY TYPE permission on a sequence can run the **ALTER TYPE** command. When separation of duties is disabled, system administrators have this permission by default. To modify the owner or schema of a type, you must be a type owner or system administrator and a member of the new owner role.

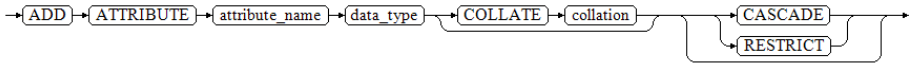
Syntax

- Modify a type.
  ALTER TYPE name action [, ...];

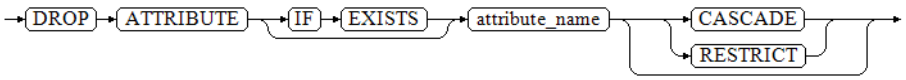
  

  The clauses corresponding to **action** are as follows:

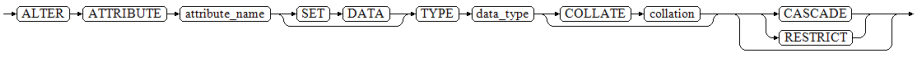
  – Add a new attribute to a composite type.
  ADD ATTRIBUTE attribute_name data_type [COLLATE collation] [CASCADE | RESTRICT]

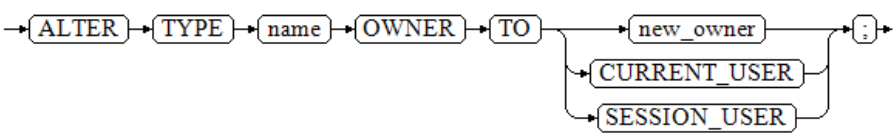
  – Delete an attribute from a composite type.
  DROP ATTRIBUTE [IF EXISTS] attribute_name [CASCADE | RESTRICT]

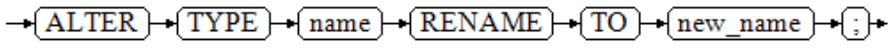
  – Change the type of an attribute in a composite type.
  ALTER ATTRIBUTE attribute_name [SET DATA] TYPE data_type [COLLATE collation]
  [CASCADE | RESTRICT]

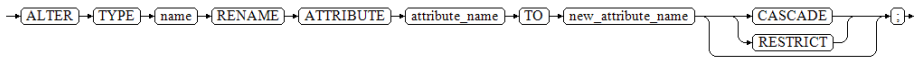
- Change the owner of a type.
  ALTER TYPE name OWNER TO { new_owner | CURRENT_USER | SESSION_USER };

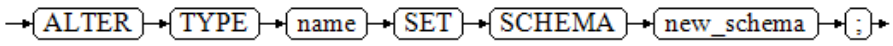
- Change the name of a type.
  ALTER TYPE name RENAME TO new_name;

- Change the name of an attribute in a composite type.
  ALTER TYPE name RENAME ATTRIBUTE attribute_name TO new_attribute_name [CASCADE | RESTRICT];

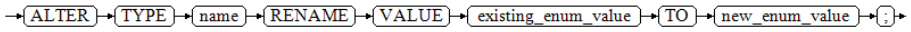


- Move a type to a new schema.
  ALTER TYPE name SET SCHEMA new_schema;



- Add a new value to an enumerated type.
  ALTER TYPE name ADD VALUE [IF NOT EXISTS] new_enum_value [{ BEFORE | AFTER } neighbor_enum_value];

- Change an enumerated value in the value list.
  ALTER TYPE name RENAME VALUE existing_enum_value TO new_enum_value;



Parameters

- **name**

  Specifies the name of an existing type that needs to be modified (optionally schema-qualified).

- **new_name**

  Specifies the new name of the type.

- **new_owner**

  Specifies the new owner of the type.

- **new_schema**

  Specifies the new schema of the type.

- **attribute_name**

  Specifies the name of the attribute to be added, modified, or deleted.

- **new_attribute_name**

  Specifies the new name of the attribute to be renamed.

- **data_type**

  Specifies the data type of the attribute to be added, or the new type of the attribute to be modified.

- **new_enum_value**

  Specifies a new enumerated value. It is a non-null string with a maximum length of 63 bytes.

- **neighbor_enum_value**

  Specifies an existing enumerated value before or after which a new enumerated value will be added.

- **existing_enum_value**

  Specifies an enumerated value to be changed. It is a non-null string with a maximum length of 63 bytes.

- **CASCADE**

  Determines that the type to be modified, its associated records, and subtables that inherit the type will all be updated.

- **RESTRICT**

  (Default) Refuses to update the associated records of the modified type.

  NOTICE

  - **ADD ATTRIBUTE**, **DROP ATTRIBUTE**, and **ALTER ATTRIBUTE** can be combined for modifying multiple attributes. For example, it is possible to add several attributes or change the types of several attributes at the same time in one command.
  - To modify a schema of a type, you must have the CREATE permission on the new schema. To change the owner, you must be a direct or indirect member of the new owning role, and the member must have the **CREATE** permission on the schema of this type. (The user can only rebuild and delete the type. However, system administrators can change ownership of any type in any way when separation of duties is disabled.) To add an attribute or modify the type of an attribute, you must also have the USAGE permission of this type.

- **CURRENT_USER**

  Specifies the current user.

- **SESSION_USER**

  Specifies the current system user.

- **COLLATE collation**

  Assigns a collation to the column, which must be a sortable data type. If the collation is not specified, the default collation for the column's data type is used.

Examples

- Modify a composite type:

```
-- Create a composite type.
gaussdb=# CREATE TYPE typ_stu AS (name varchar(10),age int);

-- Create a table and insert data into the table. The data type of the info column is typ_stu.
gaussdb=# CREATE TABLE tbl_test (id int PRIMARY KEY,info typ_stu);
gaussdb=# INSERT INTO tbl_test VALUES (1,('Jim',16));

-- View data in the tbl_test table.
gaussdb=# SELECT * FROM tbl_test;
 id |   info
----+----------
  1 | (Jim,16)
(1 row)

-- Add an attribute to a composite type.
gaussdb=# ALTER TYPE typ_stu ADD ATTRIBUTE year int CASCADE;

-- Check data in the tbl_test table again. The info column contains an extra attribute.
gaussdb=# SELECT * FROM tbl_test;
 id |   info
----+-----------
  1 | (Jim,16,)
(1 row)

-- Delete an attribute from a composite type.
gaussdb=# ALTER TYPE typ_stu DROP ATTRIBUTE year;
```


- Change the type owner.

```
-- Create a user test.
gaussdb=# CREATE ROLE test PASSWORD '********';

-- Change the owner of typ_stu to test.
gaussdb=# ALTER TYPE typ_stu OWNER TO test;

-- Query the owner of typ_stu.
gaussdb=# SELECT t1.typname, t2.rolname AS owner
FROM pg_type t1, pg_roles t2
WHERE t1.typname = 'typ_stu' AND
      t1.typowner = t2.oid;
 typname | owner
---------+--------
 typ_stu | test
(1 row)
```

- Change a type or the name of an attribute in a composite type.

```
-- Rename  attribute age of typ_stu to age1.
gaussdb=# ALTER TYPE typ_stu RENAME ATTRIBUTE age to age1;

-- Query the data whose age is 16 in the tbl_test table.
gaussdb=# SELECT id,(info).name,(info).age1 FROM tbl_test WHERE (info).age1 = 16;
 id | name | age1
----+------+------
  1 | Jim  |   16
(1 row)
```

- Move a type to a new schema.

```
-- Create a schema sctest.
gaussdb=# CREATE SCHEMA sctest;

-- Move typ_stu to the sc_test schema.
gaussdb=# ALTER TYPE typ_stu SET SCHEMA sctest;

-- Query the schema of typ_stu.
gaussdb=# \dT sctest.*
          List of data types
 Schema |      Name      | Description
--------+----------------+-------------
 sctest | sctest.typ_stu |
(1 row)

-- Drop the table.
gaussdb=# DROP TABLE tbl_test;
gaussdb=# DROP TYPE sctest.typ_stu;
gaussdb=# DROP ROLE test;
gaussdb=# DROP SCHEMA sctest;
```

- Add a new value to an enumerated type.

```
-- Create an enumerated type typ_bugstatus.
gaussdb=# CREATE TYPE typ_bugstatus AS ENUM ('create', 'modify', 'closed');

-- Create a table tbl_test1.
gaussdb=# CREATE TABLE tbl_test1 (id serial, bugstat typ_bugstatus);

-- Insert data and ensure that the value of the bugstat column must be of the enumerated type.
Otherwise, an error is reported.
gaussdb=# INSERT INTO tbl_test1 (bugstat) VALUES ('closed');
gaussdb=# INSERT INTO tbl_test1 (bugstat) VALUES ('deleted');
ERROR:  invalid input value for enum typ_bugstatus: "deleted"
LINE 1: INSERT INTO tbl_test1 (bugstat) VALUES ('deleted');
                                                ^
CONTEXT:  referenced column: bugstat

-- View data in the tbl_test1 table.
gaussdb=# SELECT * FROM tbl_test1;
 id | bugstat
----+---------
  1 | closed
```



```
(1 row)

-- Add a tag value to an enumerated type.
gaussdb=# ALTER TYPE typ_bugstatus ADD VALUE IF NOT EXISTS 'regress' BEFORE 'closed';
-- Query.
gaussdb=# \dT+ typ_bugstatus
                                 List of data types
 Schema |     Name      | Internal name | Size | Elements | Access privileges | Description
--------+---------------+---------------+------+----------+-------------------+-------------
 public | typ_bugstatus | typ_bugstatus | 4    | create  +|                   |
        |               |               |      | modify  +|                   |
        |               |               |      | regress +|                   |
        |               |               |      | closed   |                   |
(1 row)
```

- Change an enumerated value in the value list.

```
-- Change closed in typ_bugstatus to close.
gaussdb=# ALTER TYPE typ_bugstatus RENAME VALUE 'closed' TO 'close';

-- Check the data in the tbl_test1 table. closed is changed to close.
gaussdb=# SELECT * FROM tbl_test1;
 id | bugstat
----+---------
  1 | close
(1 row)

-- Query.
gaussdb=# \dT+ typ_bugstatus;
                                 List of data types
 Schema |     Name      | Internal name | Size | Elements | Access privileges | Description
--------+---------------+---------------+------+----------+-------------------+-------------
 public | typ_bugstatus | typ_bugstatus | 4    | create  +|                   |
        |               |               |      | modify  +|                   |
        |               |               |      | regress +|                   |
        |               |               |      | close    |                   |
(1 row)

-- Delete.
gaussdb=# DROP TABLE tbl_test1;
gaussdb=# DROP TYPE typ_bugstatus;
```

Helpful Links

CREATE TYPE and **DROP TYPE**

7.12.6.38 ALTER USER

Description

ALTER USER modifies the attributes of a database user.

Precautions

Session parameters modified by ALTER USER apply to a specified user and take effect in the next session.

Syntax

- Modify user permissions or other information.

```
ALTER USER user_name [ [ WITH ] option [ ... ] ];
ALTER USER user_name
    RENAME TO new_name;
ALTER USER user_name [ IN DATABASE database_name ]
    SET configuration_parameter {{ TO | = } { value | DEFAULT }|FROM CURRENT};
```



```
ALTER USER user_name
    [ IN DATABASE database_name ] RESET {configuration_parameter|ALL};
```

The option clause is as follows:

```
{ CREATEDB | NOCREATEDB }
    | { CREATEROLE | NOCREATEROLE }
    | { INHERIT | NOINHERIT }
    | { AUDITADMIN | NOAUDITADMIN }
    | { SYSADMIN | NOSYSADMIN }
    | {MONADMIN | NOMONADMIN}
    | {OPRADMIN | NOOPRADMIN}
    | {POLADMIN | NOPOLADMIN}
    | { USEFT | NOUSEFT }
    | { LOGIN | NOLOGIN }
    | { REPLICATION | NOREPLICATION }
    | {PERSISTENCE | NOPERSISTENCE}
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD { 'password' [EXPIRED] | DISABLE | EXPIRED }
    | [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY { 'password' [ REPLACE 'old_password' |
EXPIRED ] | DISABLE }
    | VALID BEGIN 'timestamp'
    | VALID UNTIL 'timestamp'
    | RESOURCE POOL 'respool'
    | USER GROUP 'groupuser'
    | PERM SPACE 'spacelimit'
    | TEMP SPACE 'tmpspacelimit'
    | SPILL SPACE 'spillspacelimit'
    | NODE GROUP logic_cluster_name
    | ACCOUNT { LOCK | UNLOCK }
    | PGUSER
```

- Change the username.

```
ALTER USER user_name
    RENAME TO new_name;
```

- Change the value of a specified parameter associated with the user.

```
ALTER USER user_name [ IN DATABASE database_name ]
    SET configuration_parameter {{ TO | = } { value | DEFAULT }|FROM CURRENT};
```

- Reset the value of a specified parameter associated with the user.

```
ALTER USER user_name
    [ IN DATABASE database_name ] RESET {configuration_parameter|ALL};
```

Parameters

- **user_name**

  Specifies the current username.

  Value range: an existing username. If a username contains uppercase letters, enclose the name with double quotation marks ("").

- **new_password**

  Specifies a new password.

  The new password must:

  – Differ from the old password.

  – Contain at least eight characters. This is the default length.

  – Differ from the username or the username spelled backward.

  – Contain at least three of the following four types of characters: uppercase characters (A to Z), lowercase characters (a to z), digits (0 to 9), and special characters, including: ~!@#$%^&*()-_=+\|[{}];:,<.>/? If the password contains characters other than the preceding characters, an error will be reported during statement execution.

  - Be enclosed by single quotation marks.

  Value range: a string

- **old_password**

  Specifies the old password.

- **ACCOUNT { LOCK | UNLOCK }**
  - **ACCOUNT LOCK**: locks an account to forbid login to databases.
  - **ACCOUNT UNLOCK**: unlocks an account to allow login to databases.
- **PGUSER**

  In the current version, the **PGUSER** attribute of a user cannot be modified.

For details about other parameters, see "Parameters" in **CREATE ROLE** and **ALTER ROLE**.

Examples

```
-- Create user jim whose login password is ********.
gaussdb=# CREATE USER jim PASSWORD '********';

-- Change the login password of user jim.
gaussdb=# ALTER USER jim IDENTIFIED BY '**********' REPLACE '********';

-- Set enable_seqscan to on. (The setting will take effect in the next session.)
gaussdb=# ALTER USER jim SET enable_seqscan TO on;

-- Reset the enable_seqscan parameter for jim.
gaussdb=# ALTER USER jim RESET enable_seqscan;

-- Lock jim.
gaussdb=# ALTER USER jim ACCOUNT LOCK;

-- Unlock jim.
gaussdb=# ALTER USER jim ACCOUNT UNLOCK;

-- Change the username.
gaussdb=# ALTER USER jim RENAME TO lisa;

-- Delete the user.
gaussdb=# DROP USER lisa CASCADE;
```

Helpful Links

CREATE ROLE, **CREATE USER**, and **DROP USER**

7.12.6.39 ALTER USER MAPPING

Description

ALTER USER MAPPING changes the definition of the mapping from a user to a foreign server. The owner of the foreign server can change the user mapping of the server for any user. In addition, if the USAGE permission on the server has been granted to a user, the user can change the user mapping of its own username.

Precautions

- If the **password** option is displayed, ensure that the **usermapping.key.cipher** and **usermapping.key.rand** files exist in the *$GAUSSHOME*/**bin** directory of

each node in GaussDB. If the two files do not exist, use the **gs_guc** tool to generate them and use the **gs_ssh** tool to release them to the *$GAUSSHOME*/**bin** directory on each node. For details, see the description in **OPTIONS**.

- When multi-layer quotation marks are used for sensitive columns (such as **password**) in OPTIONS, the semantics is different from that in the scenario where quotation marks are not used. Therefore, sensitive columns are not identified for anonymization.

Syntax

```
ALTER USER MAPPING FOR { user_name | USER | CURRENT_USER | PUBLIC }
    SERVER server_name
    OPTIONS ( [ ADD | SET | DROP ] option ['value'] [, ... ] );
```

In **OPTIONS**, **ADD**, **SET**, and **DROP** are operations to be performed. If these operations are not specified, **ADD** operations will be performed by default. **option** and **value** are the parameters and values of the corresponding operation.

Parameters

- **user_name**

  Specifies username of the mapping.

  CURRENT_USER and **USER** match the name of the current user. **PUBLIC** is used to match all current and future usernames in the system.

- **server_name**

  Specifies name of the server to which the user is mapped.

- **OPTIONS**

  Changes an option for the user mapping. The new option overwrites any previously specified option. **ADD**, **SET**, and **DROP** are operations to be performed. If the operation is not set explicitly, **ADD** is used. The option name must be unique and will be validated with the foreign data wrapper of the server.

  NOTE

  - User passwords are encrypted and stored in the system catalog **PG_USER_MAPPING**. During the encryption, **usermapping.key.cipher** and **usermapping.key.rand** are used as the encryption password file and encryption factor. Before using the tool for the first time, create the two files, save the files to the *$GAUSSHOME*/**bin** directory on each node, and ensure that you have the read permission on the files. **gs_ssh** helps you quickly place files in the specified directory of each node.
    ```
    gs_ssh -c "gs_guc generate -o usermapping -S default -D $GAUSSHOME/bin"
    ```
  - If the **-S** parameter is set to default, a password is randomly generated. You can also specify a password for the **-S** parameter to ensure the security and uniqueness of the generated password file. You do not need to save or memorize the password. For details about other parameters, see the description of the gs_guc tool in the *Tool Reference*.

Examples

```
-- Create a role.
gaussdb=# CREATE ROLE bob PASSWORD '********';

-- Create a foreign server.
```



```
gaussdb=# CREATE SERVER my_server FOREIGN DATA WRAPPER log_fdw;

-- Create a user mapping.
gaussdb=# CREATE USER MAPPING FOR bob SERVER my_server OPTIONS (user 'bob', password '********');

-- Modify the user mapping.
gaussdb=# ALTER USER MAPPING FOR bob SERVER my_server OPTIONS (SET password '********');

-- Delete the user mapping.
gaussdb=# DROP USER MAPPING FOR bob SERVER my_server;

-- Delete the foreign server.
gaussdb=# DROP SERVER my_server;

-- Delete the role.
gaussdb=# DROP ROLE bob;
```

Helpful Links

CREATE USER MAPPING and **DROP USER MAPPING**

7.12.6.40 ALTER VIEW

Description

ALTER VIEW changes the auxiliary attributes of a view. If you want to change the query definition of a view, use CREATE OR REPLACE VIEW.

Precautions

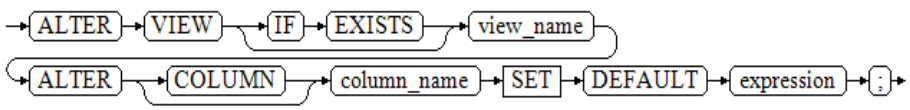
Only the view owner or a user granted the ALTER permission on a view can run the **ALTER VIEW** command. When separation of duties is disabled, a system administrator has this permission by default. The following are permission constraints depending on the attributes to be modified:

- To modify the schema of a view, you must be the owner of the view or system administrator and have the CREATE permission on the new schema. When separation of duties is enabled, a system administrator cannot change the view mode.
- To modify the owner of a view, you must be the owner of the view or system administrator and a member of the new owner role, with the CREATE permission on the schema of the view. When separation of duties is enabled, a system administrator cannot change the owner of a view.
- Do not change the type of a column in a view.

Syntax

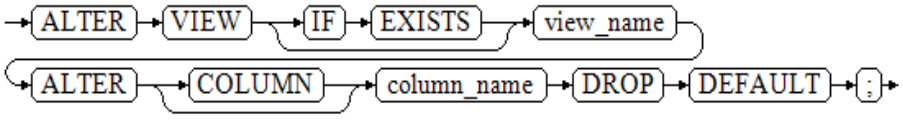
- Set the default value of a view column.

```
ALTER VIEW [ IF EXISTS ] view_name
    ALTER [ COLUMN ] column_name SET DEFAULT expression;
```



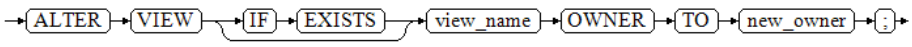
- Remove the default value of a view column.

```
ALTER VIEW [ IF EXISTS ] view_name
    ALTER [ COLUMN ] column_name DROP DEFAULT;
```

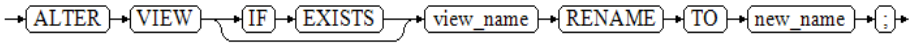
- Change the owner of a view.

```
ALTER VIEW [ IF EXISTS ] view_name
    OWNER TO new_owner;
```



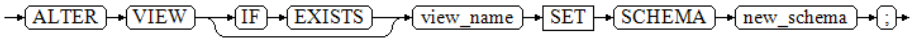
- Rename a view.

```
ALTER VIEW [ IF EXISTS ] view_name
    RENAME TO new_name;
```



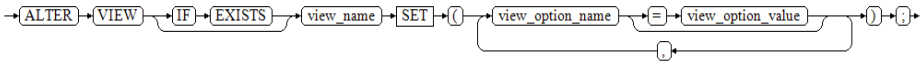
- Set the schema of a view.

```
ALTER VIEW [ IF EXISTS ] view_name
    SET SCHEMA new_schema;
```



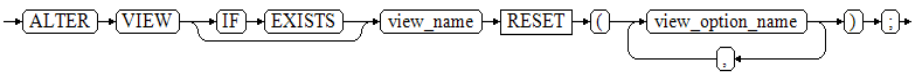
- Set the options of a view.

```
ALTER VIEW [ IF EXISTS ] view_name
    SET ( { view_option_name [ = view_option_value ] } [, ... ] );
```



- Reset the options of a view.

```
ALTER VIEW [ IF EXISTS ] view_name
    RESET ( view_option_name [, ... ] );
```



Parameters

- **IF EXISTS**

  If this option is used, no error is generated when the view does not exist, and only a message is displayed.

- **view_name**

  Specifies the view name, which can be schema-qualified.

  Value range: a string. It must be an existing view name.

- **column_name**

  Specifies the column name.

  Value range: a string. It must be an existing view column name.

- **SET/DROP DEFAULT**

  Sets or deletes the default value of a column. This parameter does not take effect.

- **new_owner**

  Specifies the new owner of a view.

- **new_name**

  Specifies the new view name.

- **new_schema**

  Specifies the new schema of the view.

- **view_option_name [= view_option_value]**

  Specifies an optional parameter for a view.

  - **security_barrier**: specifies whether the view provides row-level security. The value is of the Boolean type. The default value is **true**.
  - **check_option**: controls the behavior of updating a view. This parameter can be set to **CASCADED** or **LOCAL**. This parameter cannot be left blank.

- **expression**

  Specifies constants, functions, or SQL expressions.

Examples

- Rename a view.

```
-- Create the test_tbl table.
gaussdb=# CREATE TABLE test_tb1(col1 INT,col2 INT);

-- Create a view.
gaussdb=# CREATE VIEW abc AS SELECT * FROM test_tb1;

-- Rename the view.
gaussdb=# ALTER VIEW IF EXISTS abc RENAME TO test_v1;

-- Query the view.
gaussdb=# \dv
            List of relations
 Schema |  Name   | Type | Owner | Storage
--------+---------+------+-------+---------
 public | test_v1 | view | omm   |
(1 row)
```

- Change the owner of the view.

```
-- Create a user.
gaussdb=# CREATE ROLE role_test PASSWORD '********';

-- Change the owner of the view.
gaussdb=# ALTER VIEW IF EXISTS test_v1 OWNER TO role_test;

-- Query the view information.
gaussdb=# \dv
              List of relations
 Schema |  Name   | Type |   Owner   | Storage
--------+---------+------+-----------+---------
 public | test_v1 | view | role_test |
(1 row)
```

- Set the schema of the view.

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tcpds;

-- Change the schema of the view.
gaussdb=# ALTER VIEW test_v1 SET SCHEMA tcpds;

-- Query the view information.
gaussdb=# \dv tcpds.test_v1;
              List of relations
 Schema |  Name   | Type |   Owner   | Storage
--------+---------+------+-----------+---------
 tcpds  | test_v1 | view | role_test |
(1 row)
```

- Set and reset view options.

```
-- Modify view options.
gaussdb=# ALTER VIEW tcpds.test_v1 SET (security_barrier = TRUE);
```



```
ALTER VIEW

-- Query.
gaussdb=# \dv tcpds.test_v1;
                  List of relations
 Schema |  Name   | Type | Owner  |         Storage
--------+---------+------+--------+-------------------------
 tcpds  | test_v1 | view | chenxi | {security_barrier=true}
(1 row)

-- Modify the check_option option.
gaussdb=# ALTER VIEW tcpds.test_v1 SET (check_option = 'LOCAL');
ALTER VIEW

-- Query.
gaussdb=# \dv tcpds.test_v1;
                           List of relations
 Schema |  Name   | Type | Owner  |                  Storage
--------+---------+------+--------+-----------------------------------------------
 tcpds  | test_v1 | view | chenxi | {security_barrier=true,check_option=LOCAL}
(1 row)

-- Reset view options.
gaussdb=# ALTER VIEW tcpds.test_v1 RESET (security_barrier);
ALTER VIEW
gaussdb=# ALTER VIEW tcpds.test_v1 RESET (check_option);
ALTER VIEW

-- Query.
gaussdb=# \dv tcpds.test_v1;
            List of relations
 Schema |  Name   | Type | Owner  | Storage
--------+---------+------+--------+---------
 tcpds  | test_v1 | view | chenxi |
(1 row)

-- Delete the test_v1 view.
gaussdb=# DROP VIEW tcpds.test_v1;
DROP VIEW

-- Delete the test_tb1 table.
gaussdb=# DROP TABLE test_tb1;
DROP TABLE

-- Delete the user.
gaussdb=# DROP ROLE role_test;
DROP ROLE

-- Delete the schema.
gaussdb=# DROP SCHEMA tcpds;
DROP SCHEMA
```

Helpful Links

CREATE VIEW and **DROP VIEW**

7.12.6.41 ANALYZE | ANALYSE

Description

- ANALYZE collects statistics about ordinary tables in a database and stores the results in the PG_STATISTIC and PG_STATISTIC_EXT system catalogs. After you run the **ANALYZE** command, you can query the collected statistics in the system catalogs or in the system views PG_STATS and PG_EXT_STATS. The execution plan generator uses these statistics to generate the most effective execution plan.

- If no parameters are specified, ANALYZE analyzes each table and partitioned table in the database. You can also specify **table_name**, **column_name**, and **partition_name** to limit the analysis to a specified table, column, or partitioned table.
- ANALYZE | ANALYSE VERIFY can be used to check whether data files of ordinary tables in a database are damaged.
- Statistics collected each time are stored in historical statistics tables (**GS_STATISTIC_HISTORY**, **GS_STATISTIC_EXT_HISTORY**, and **GS_TABLESTATS_HISTORY**). The number of historical tables and the retention period of statistics are specified by the **stats_history_record_limit** and **stats_history_retention_time** parameters.

Precautions

- Non-temporary tables cannot be analyzed in an anonymous block, transaction block, function, or stored procedure. Temporary tables can be analyzed during stored procedures, but statistics cannot be rolled back.
- If remote read is not involved, the remote read parameter does not take effect. If a key system catalog page is damaged, an error is reported and the detection stops.
- If no parameter to be analyzed is specified, tables on which the current user has the corresponding permission are analyzed by default. If a table parameter is specified, only the specified table is analyzed.
- To perform ANALYZE operation to a table, you must be a table owner or a user granted the VACUUM permission on the table. By default, a system administrator has this permission. However, database owners are allowed to analyze all tables in their databases, except shared catalogs. (The restriction for shared catalogs means that a true database-wide analyze operation can only be executed by the system administrator). ANALYZE skips tables on which users do not have permissions.
- ANALYZE does not collect columns for which comparison or equivalent operations cannot be performed, for example, columns of the cursor type.

Syntax

- Collect statistics information about a table.

```
{ ANALYZE | ANALYSE } [ VERBOSE ]
    [ table_name [ ( column_name [, ...] ) ] ] [ WITH PARTITION_MODE ];
```

- Collect partition statistics information about a partitioned table.

```
{ ANALYZE | ANALYSE } [ VERBOSE ]
    table_name [ ( column_name [, ...] ) ] { SUBPARTITION | PARTITION } ( partition_name ) ;
```

  NOTE

  If the keyword PARTITION is used, **partition_name** must be the name of a level-1 partition. If the keyword SUBPARTITION is used, **partition_name** must be the name of a level-2 partition.

- Collect statistics about multiple columns manually.

```
{ ANALYZE | ANALYSE } [ VERBOSE ]
    table_name (( column_1_name, column_2_name [, ...] )) [ WITH partition_mode ];
```


NOTE

- If the GUC parameter **enable_functional_dependency** is disabled, the statistics about a maximum of 32 columns can be collected at a time. If the GUC parameter **enable_functional_dependency** is enabled, the statistics about a maximum of 4 columns can be collected at a time.
- You are not allowed to collect statistics about multiple columns in system catalogs and global temporary tables.

- Collect statistics about multiple columns automatically.

  After the **auto_statistic_ext_columns** parameter is enabled and the ANALYZE statement is executed, multi-column statistics are automatically created based on the index prefix of the table. The number of columns in the multi-column statistics cannot exceed the value of **auto_statistic_ext_columns**.

  For example, if index (a, b, c, d) exists in table **t** and **auto_statistic_ext_columns** is set to **4**, multi-column statistics about (a, b), (a, b, c), and (a, b, c, d) are created after table **t** is analyzed.

  ```
  { ANALYZE | ANALYSE } [ VERBOSE ] table_name [ WITH partition_mode ];
  ```

- Check the data files in the current database.

  ```
  { ANALYZE | ANALYSE } VERIFY { FAST | COMPLETE };
  ```

  NOTE

  - In fast mode, DML operations need to be performed on the tables to be verified concurrently. As a result, an error is reported during the verification. In the current fast mode, data is directly read from the disk. When other threads modify files concurrently, the obtained data is incorrect. Therefore, you are advised to perform the verification offline.
  - You can perform operations on the entire database. Because a large number of tables are involved, you are advised to save the results in redirection mode.

    ```
    gsql -d database -p port -f sqlfile> sqllog.txt  2>&1
    ```

  - NOTICE is displayed only for externally visible tables. The detection of internal tables is included in the external tables on which the internal tables depend and is not displayed externally.
  - This statement can be executed with error tolerance.
  - If a key system catalog is damaged during a full database operation, an error is reported and the operation stops.

- Check data files of tables and indexes.

  ```
  { ANALYZE | ANALYSE } VERIFY { FAST | COMPLETE } { table_name | index_name } [CASCADE];
  ```

  NOTE

  - Operations on ordinary tables and index tables are supported, but CASCADE operations on indexes of index tables are not supported. The CASCADE mode is used to process all index tables of the main table. When the index tables are checked separately, the CASCADE mode is not required.
  - When the main table is checked, the internal tables of the main table, such as the TOAST table, are also checked.
  - When the system displays a message indicating that the index table is damaged, you are advised to run the **reindex** command to rebuild the index.

- Check the data files of the table partition.

  ```
  { ANALYZE | ANALYSE } VERIFY { FAST | COMPLETE } table_name PARTITION (patrition_name) [CASCADE];
  ```


NOTE

- You can check a single partition of a table, but cannot perform the CASCADE operation on the indexes of an index table.
- Temporary tables and unlogged tables are not supported.

Parameters

- **VERBOSE**

  Enables the display of progress messages.

  NOTE

  If **VERBOSE** is specified, ANALYZE displays the progress information, indicating the table that is being processed. Statistics about tables are also displayed.

- **table_name**

  Specifies the name (possibly schema-qualified) of a specific table to analyze. If omitted, all regular tables (but not foreign tables) in the current database are analyzed.

  Currently, you can use ANALYZE to collect statistics only from row-store tables.

  Value range: an existing table name.

- **column_name**, **column_1_name**, **column_2_name**

  Specifies the name of a specific column to analyze. All columns are analyzed by default.

  Value range: An existing column name.

- **partition_name**

  Assumes that the table is a partitioned table. You can specify **partition_name** following the keyword PARTITION to analyze the statistics of this table.

  Value range: a partition name of a table.

- **index_name**

  Specifies the name of the specific index table to be analyzed (possibly schema-qualified).

  Value range: an existing table name.

- **FAST|COMPLETE**

  The **FAST** mode verifies the CRC and page header of the table. If the verification fails, an alarm is generated. In **COMPLETE** mode, the pointer and tuple of the table are parsed and verified.

- **CASCADE**

  In **CASCADE** mode, all indexes of the current table are verified.

- **PARTITION_MODE**

  Cascadingly collects statistics about partitioned tables. The following table describes the options and their meanings. This is not applicable to non-partitioned tables.

Table 7-216 PARTITION_MODE options

PARTITION_MODE Option	Description
ALL	Statistics about the entire table, level-1 partitions, and level-2 partitions are collected.
GLOBAL	Statistics about the entire table are collected.
PARTITION	Statistics about the level-1 partition are collected.
GLOBAL AND PARTITION	Statistics about the entire table and level-1 partition are collected.
SUBPARTITION	Statistics about the level-2 partition are collected.
ALL COMPLETE	Statistics about the entire table, level-1 partitions, and level-2 partitions are collected.
AUTO	Default value. The parameter configured in **statistic_granularity** or **default_statistic_granularity** prevails. The priority of the table-level parameter **statistic_granularity** is higher than that of the global parameter **default_statistic_granularity**.

A difference between **ALL** and **ALL COMPLETE** is as follows: **ALL_COMPLETE** uses a higher sampling rate and takes a longer time to calculate statistics.

CAUTION

- When a version earlier than 505.0.0 is upgraded to 505.0.0 or later, **PARTITION_MODE** does not take effect during the upgrade observation period. The behavior of **PARTITION_MODE** is the same as that of the source version.
- When the entire database is analyzed, the partition mode cannot be specified. That is, syntax such as ANALYZE WITH GLOBAL is not supported.
- When the entire database is analyzed, **default_statistic_granularity** can be set to **GLOBAL** or **ALL**. In this case, the ANALYZE behavior is the same as that defined in **default_statistic_granularity**. When **default_statistic_granularity** is set to other values, the ANALYZE behavior is degraded to the GLOBAL behavior.

Examples

- Collect statistics information about a table.

```
-- Create the customer_info table.
gaussdb=# CREATE TABLE customer_info(
wr_returned_date_sk INTEGER ,
wr_returned_time_sk INTEGER ,
wr_item_sk INTEGER NOT NULL
,wr_refunded_customer_sk INTEGER );

-- Run ANALYZE to update statistics.
gaussdb=# ANALYZE customer_info;

-- Run ANALYZE VERBOSE to update statistics and display information about the customer_info table.
gaussdb=# ANALYZE VERBOSE customer_info;
INFO:  analyzing "public.customer_info"(datanode pid=38661)
INFO:  ANALYZE INFO : estimate total rows of "customer_info": scanned 0 pages of total 0 pages with 1 retry times, containing 0 live rows and 0 dead rows,  estimated 0 total rows(datanode pid=38661)
INFO:  ANALYZE INFO : "customer_info": scanned 0 of 0 pages, containing 0 live rows and 0 dead rows; 0 rows in sample, 0 estimated total rows(datanode pid=38661)

-- Run ANALYZE VERBOSE to export the wr_returned_time_sk column information in the customer_info table.
gaussdb=# ANALYZE VERBOSE customer_info(wr_returned_time_sk);
INFO:  analyzing "public.customer_info"(datanode pid=38661)
INFO:  ANALYZE INFO : estimate total rows of "customer_info": scanned 0 pages of total 0 pages with 1 retry times, containing 0 live rows and 0 dead rows,  estimated 0 total rows(datanode pid=38661)
INFO:  ANALYZE INFO : "customer_info": scanned 0 of 0 pages, containing 0 live rows and 0 dead rows; 0 rows in sample, 0 estimated total rows(datanode pid=38661)
```

- Collect statistics about a partitioned table.

```
-- Create a partitioned table.
gaussdb=# CREATE TABLE customer_par(
wr_returned_date_sk INTEGER ,
wr_returned_time_sk INTEGER ,
wr_item_sk INTEGER NOT NULL,
wr_returned_customer_sk INTEGER)
PARTITION BY RANGE(wr_returned_date_sk)(PARTITION P1 VALUES LESS THAN(2452275),PARTITION P2 VALUES LESS THAN(2452640),PARTITION P3 VALUES LESS THAN(2453000),PARTITION P4 VALUES LESS THAN(MAXVALUE))ENABLE ROW MOVEMENT;

-- Run ANALYZE to update statistics.
gaussdb=# ANALYZE customer_par;

-- Run ANALYZE VERBOSE to export the information in the customer_par table.
gaussdb=# ANALYZE VERBOSE customer_par;
INFO:  analyzing "public.customer_par"(datanode pid=38661)
-- Run ANALYZE VERBOSE to output information about the P1 level-1 partitioned table.
gaussdb=# ANALYZE VERBOSE customer_par PARTITION(P1);
INFO:  analyzing "public.customer_par"(datanode pid=38661)
```

- Collect statistics about multiple columns manually.

```
-- Manually collect statistics in the wr_returned_date_sk and wr_returned_time_sk columns.
gaussdb=# ANALYZE VERBOSE customer_info (wr_returned_date_sk,wr_returned_time_sk);
INFO:  analyzing "public.customer_info"(datanode pid=38661)
INFO:  ANALYZE INFO : estimate total rows of "customer_info": scanned 0 pages of total 0 pages with 1 retry times, containing 0 live rows and 0 dead rows,  estimated 0 total rows(datanode pid=38661)
INFO:  ANALYZE INFO : "customer_info": scanned 0 of 0 pages, containing 0 live rows and 0 dead rows; 0 rows in sample, 0 estimated total rows(datanode pid=38661)
```

- Collect statistics about multiple columns automatically.

```
-- Create an index for the customer_info table.
gaussdb=# CREATE INDEX customer_index ON customer_info USING btree(wr_returned_date_sk,wr_returned_time_sk,wr_item_sk,wr_refunded_customer_sk);

-- Set auto_statistic_ext_columns to 4.
gaussdb=# set auto_statistic_ext_columns=4;

-- Collect statistics about multiple columns automatically.
```



```
gaussdb=# ANALYZE VERBOSE customer_info;
INFO:  analyzing "public.customer_info"(datanode pid=38661)
INFO:  ANALYZE INFO : estimate total rows of "customer_info": scanned 0 pages of total 0 pages with 1 retry times, containing 0 live rows and 0 dead rows,  estimated 0 total rows(datanode pid=38661)
INFO:  ANALYZE INFO : "customer_info": scanned 0 of 0 pages, containing 0 live rows and 0 dead rows; 0 rows in sample, 0 estimated total rows(datanode pid=38661)
```

- Check the data files in the current database.

```
gaussdb=# ANALYZE VERIFY FAST;
```

- Check data files of tables and indexes.

```
-- Check the customer_info table.
gaussdb=# ANALYZE VERIFY FAST customer_info;

-- Check the customer_index index.
gaussdb=# ANALYZE VERIFY FAST customer_index;
```

- Check the data files of the table partition.

```
-- Check the P1 partition in the customer_par partitioned table.
gaussdb=# ANALYZE VERIFY FAST customer_par PARTITION (P1);
```

- Delete data.

```
-- Delete the index customer_index.
gaussdb=# DROP INDEX customer_index;

-- Drop the customer_info table.
gaussdb=# DROP TABLE customer_info;

-- Drop the customer_par partitioned table.
gaussdb=# DROP TABLE customer_par;
```

7.12.7 B

7.12.7.1 BEGIN

Description

BEGIN may be used to initiate an anonymous block or a single transaction.

An anonymous block is a structure that can dynamically create and execute stored procedure code instead of permanently storing code as a database object in the database.

Precautions

None

Syntax

- Enable an anonymous block.

```
[DECLARE [declare_statements]]
BEGIN
execution_statements
END;
/
```

- Start a transaction.

```
BEGIN [ WORK | TRANSACTION ]
 [
  {
    ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED | SERIALIZABLE | REPEATABLE READ }
    | { READ WRITE | READ ONLY }
```



```
    } [, ...]
];
```

Parameters

- **declare_statements**

  Declares a variable, including its name and type, for example, **sales_cnt int**.

- **execution_statements**

  Specifies the statement to be executed in an anonymous block.

  Value range: DML operations (such as SELECT, INSERT, DELETE, and UPDATE) or registered functions in the system catalog.

- **WORK | TRANSACTION**

  Specifies the optional keyword in the BEGIN syntax, which is meaningless.

- **ISOLATION LEVEL**

  Specifies the transaction isolation level that determines the data that a transaction can view if other concurrent transactions exist.

  NOTE

  The isolation level of a transaction cannot be reset after the first clause (INSERT, DELETE, UPDATE, FETCH, or COPY) for modifying data is executed in the transaction.

  Value range:

  - **READ COMMITTED**: Only submitted data is read. This is the default action.
  - **READ UNCOMMITTED**: Committed data is probably read. This isolation level is provided to handle CN breakdown emergencies. On this isolation level, you are advised to only read data to prevent inconsistency.
  - **REPEATABLE READ**: Only the data committed before transaction start is read. Uncommitted data or data committed in other concurrent transactions cannot be read.
  - **SERIALIZABLE**: Currently, this isolation level is not supported. Setting this isolation level is equivalent to **REPEATABLE READ**.

- **READ WRITE | READ ONLY**

  Specifies the transaction access mode (read/write or read only).

Examples

- Start a transaction.

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE tbl_test1(col1 int, col2 int);
gaussdb=# INSERT INTO tbl_test1 VALUES (1,1), (2,2), (3,3);

-- Start a transaction in default mode.
gaussdb=# BEGIN;
INSERT INTO tbl_test1 VALUES (4,4);
END;

-- Start a transaction with the separation level being REPEATABLE READ.
gaussdb=# BEGIN ISOLATION LEVEL REPEATABLE READ;
SELECT * FROM tbl_test1;
END;

-- Drop the table.
gaussdb=# DROP TABLE tbl_test1;
```


- Use anonymous blocks.

```
-- Generate a string using anonymous blocks.
gaussdb=# BEGIN
dbe_output.print_line('Hello');
END;
/
```

Helpful Links

START TRANSACTION

7.12.8 C

7.12.8.1 CALL

Description

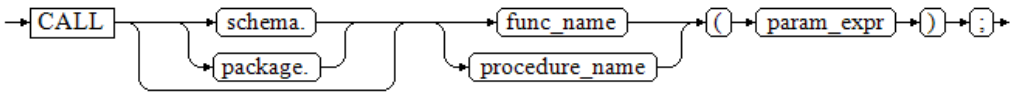
Calls defined functions and stored procedures.

Precautions

The owner of a function or stored procedure, users granted with the EXECUTE permission on the function or stored procedure, or users granted with the EXECUTE ANY FUNCTION permission can call the function or stored procedure. When separation of duties is disabled, a system administrator has the permission to use the CALL commands by default.

Syntax

```
CALL [schema.|package.] {func_name| procedure_name} ( param_expr );
```



Parameters

- **schema**

  Specifies the name of the schema where a function or stored procedure is located.

- **package**

  Specifies the name of the package where a function or stored procedure is located.

- **func_name**

  Specifies the name of the function or stored procedure to be called.

  Value range: an existing role name

  NOTE

  You can use database links to perform operations on remote functions or stored procedures. For details, see DATABASE LINK.

- **param_expr**

Specifies a list of parameters. Use := or => to separate a parameter name and its value. This method allows parameters to be placed in any order. If only parameter values are in the list, the value order must be the same as that defined in the function or stored procedure.

Value range: an existing function parameter name or stored procedure parameter name

NOTE

The parameters include input parameters (whose name and type are separated by **IN**) and output parameters (whose name and type are separated by **OUT**). When you run the **CALL** command to call a function or stored procedure, the parameter list must contain an output parameter for non-overloaded functions. You can set the output parameter to a variable or any constant. For details, see **Examples**. For an overloaded package function, the parameter list can have no output parameter, but the function may not be found. If an output parameter is contained, it must be a constant.

Examples

```
-- Create the func_add_sql function, calculate the sum of two integers, and return the result.
gaussdb=# CREATE FUNCTION func_add_sql(num1 integer, num2 integer) RETURN integer
AS
BEGIN
RETURN num1 + num2;
END;
/

-- Transfer by parameter value.
gaussdb=# CALL func_add_sql(1, 3);

-- Transfer by naming tag method.
gaussdb=# CALL func_add_sql(num1 => 1,num2 => 3);
gaussdb=# CALL func_add_sql(num2 := 2, num1 := 3);

-- Delete the function.
gaussdb=# DROP FUNCTION func_add_sql;

-- Create a function with output parameters.
gaussdb=# CREATE FUNCTION func_increment_sql(num1 IN integer, num2 IN integer, res OUT integer)
RETURN integer
AS
BEGIN
res := num1 + num2;
END;
/

-- Transfer a constant as an output parameter.
gaussdb=# CALL func_increment_sql(1,2,1);

-- Delete the function.
gaussdb=# DROP FUNCTION func_increment_sql;
```

Helpful Links

CREATE FUNCTION and **CREATE PROCEDURE**

7.12.8.2 CHECKPOINT

Function

A checkpoint is a point in the transaction log sequence at which all data files have been updated to reflect the information in the log. All data files will be flushed to a disk.

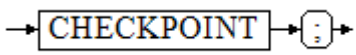
It sets transaction log checkpoints. By default, WALs periodically specify checkpoints in a transaction log. You may use gs_guc to specify run-time parameters **checkpoint_segments**, **checkpoint_timeout**, and **incremental_checkpoint_timeout** to adjust the atomized checkpoint intervals.

Precautions

- Only the system administrator and O&M administrator can invoke CHECKPOINT.
- CHECKPOINT forces an immediate checkpoint when the related command is issued, without waiting for a regular checkpoint scheduled by the system.

Syntax

```
CHECKPOINT;
```



Parameter Description

None

Examples

```
-- Set a checkpoint.
gaussdb=# CHECKPOINT;
```

7.12.8.3 CLEAN CONNECTION

Description

Clears database connections. You may use this statement to delete a specific user's connections to a specified database.

Precautions

- GaussDB does not support specified nodes and supports only TO ALL.
- This function can be used to clear the normal connections that are being used only in force mode.

Syntax

```
CLEAN CONNECTION
    TO { COORDINATOR ( nodename [, ... ] ) | NODE ( nodename [, ... ] )| ALL [ CHECK ] [ FORCE ] }
    [ FOR DATABASE dbname ]
    [ TO USER username ];
```


Parameters

- **CHECK**

  This parameter can be specified only when the node list is specified as **TO ALL**. Setting this parameter will check whether a database is accessed by other sessions before its connections are cleared. If any sessions are detected before **DROP DATABASE** is executed, an error will be reported and the database will not be deleted.

- **FORCE**

  This parameter can be specified only when the node list is specified as **TO ALL**. Setting this parameter will send **SIGTERM** signals to all the threads related to the specified **dbname** and **username** and forcibly shut them down.

- **COORDINATOR (nodename [, ...]) | NODE (nodename [, ...]) | ALL**

  Only **TO ALL** is supported. This parameter must be specified. All specified connections on the node will be deleted.

- **dbname**

  Deletes connections to a specified database. If this parameter is not specified, connections to all databases will be deleted.

  Value range: an existing database name

- **username**

  Deletes connections of a specific user. If this parameter is not specified, connections of all users will be deleted.

  Value range: an existing username

Examples

```
-- Create the test_clean_connection database.
gaussdb=# CREATE DATABASE test_clean_connection;

-- Create user jack.
gaussdb=# CREATE USER jack PASSWORD '********';

-- Log in to the database as the user jack in another session, and query the connection information in the view.
gaussdb=# SELECT datname,usename,application_name,waiting,state
          FROM pg_stat_activity
          WHERE datname = 'test_clean_connection';
       datname        | usename | application_name | waiting | state
----------------------+---------+------------------+---------+-------
 test_clean_connection | jack    | gsql             | f       | idle
(1 row)

-- If you directly delete the test_clean_connection database, the following error information is displayed:
gaussdb=# DROP DATABASE test_clean_connection;
ERROR:  Database "test_clean_connection" is being accessed by other users. You can stop all connections by command: "clean connection to all force for database XXXX;" or wait for the sessions to end by querying view: "pg_stat_activity".
DETAIL:  There is 1 other session using the database.

-- Delete the connections to all nodes for logging in to the test_clean_connection database.
-- If the FORCE parameter is not used, connections in other states cannot be deleted.
gaussdb=# CLEAN CONNECTION TO ALL FORCE FOR DATABASE test_clean_connection;

-- Query the connection for logging in to the test_clean_connection database.
gaussdb=# SELECT datname,usename,application_name,waiting,state
          FROM pg_stat_activity
          WHERE datname = 'test_clean_connection';
 datname | usename | application_name | waiting | state
```



```
---------+---------+------------------+---------+-------
(0 rows)


-- Delete the test_clean_connection database.
gaussdb=# DROP DATABASE test_clean_connection;

-- Delete user jack.
gaussdb=# DROP USER jack;
```

7.12.8.4 CLOSE

Function

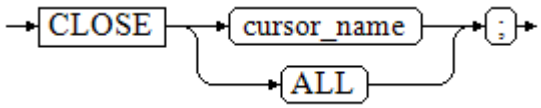
CLOSE frees the resources associated with an open cursor.

Precautions

- After a cursor is closed, no subsequent operations are allowed on it.
- A cursor should be closed when it is no longer needed.
- Every non-holdable open cursor is implicitly closed when a transaction is terminated by COMMIT or ROLLBACK.
- A holdable cursor is implicitly closed if the transaction that created it aborts by ROLLBACK.
- If the cursor creation transaction is successfully committed, the holdable cursor remains open until an explicit CLOSE operation is executed, or the client disconnects.
- GaussDB does not have an explicit OPEN cursor statement. A cursor is considered open when it is declared. You can view all available cursors by querying the pg_cursors system view.

Syntax

```
CLOSE { cursor_name | ALL };
```



Parameter Description

- **cursor_name**

  Specifies the name of a cursor to be closed.
- **ALL**

  Closes all open cursors.

Examples

See **Examples** in section "FETCH."

Helpful Links

FETCH and **MOVE**

7.12.8.5 CLUSTER

Description

- Clusters a table based on an index.
- CLUSTER instructs GaussDB to cluster the table specified by **table_name** based on the index specified by **index_name**. The index must have been defined by **table_name**.
- When a table is clustered, it is physically reordered based on the index information. Clustering is a one-time operation. When the table is subsequently updated, the changes are not clustered. That is, no attempt is made to store new or updated rows according to their index order.
- When a table is clustered, GaussDB records which index the table was clustered by. CLUSTER table_name reclusters the clustered index that was previously recorded in the table. You can also use ALTER TABLE table_name CLUSTER on index_name to set the index of a specified table for subsequent CLUSTER operations, or use ALTER TABLE table_name SET WITHOUT CLUSTER to clear the previously clustered index of a specified table.
- If CLUSTER does not contain parameters, all tables that have been clustered in the database owned by the current user will be reprocessed. If a system administrator uses this command, all clustered tables are reclustered.
- When a table is clustered, an ACCESS EXCLUSIVE lock is requested on the table. This avoids that other operations (including read and write operations) are performed on the table before the CLUSTER operation is complete.

Precautions

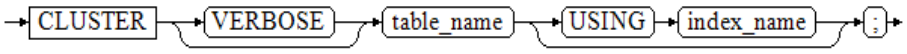
- Only row-store B-tree indexes support CLUSTER.
- In the case where you are accessing single rows randomly within a table, the actual order of the data in the table is unimportant. However, if there are many accesses to some data and an index groups the data, using the CLUSTER index improves performance.
- If you use an index to query a table for a range or multiple rows, CLUSTER will also help because once the index identifies the table page for the first row that matches, all other rows that match are probably already on the same table page. This saves disk accesses and speeds up the query.
- During clustering, the system creates a temporary backup of the table created in the index sequence and a temporary backup of each index in the table. Therefore, ensure that the disk has sufficient free space during clustering, which is at least the sum of the table size and all index sizes.
- CLUSTER records which indexes have been used for clustering. Therefore, you can manually specify indexes for the first time, cluster specified tables, and set a maintenance script that will be executed periodically. You only need to run the **CLUSTER** command without parameters. In this way, tables that you want to periodically cluster can be automatically updated.
- The optimizer records table clustering statistics. After clustering a table, you need to execute the ANALYZE operation to ensure that the optimizer has the latest clustering information. Otherwise, the optimizer may select a non-optimal query plan.
- CLUSTER cannot be executed in transactions.

- If the GUC parameter **xc_maintenance_mode** is not set to **on**, the CLUSTER operation skips all system catalogs.

Syntax

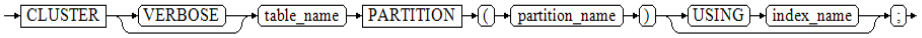
- Cluster a table.

```
CLUSTER [ VERBOSE ] table_name [ USING index_name ];
```



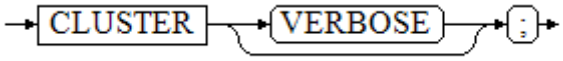
- Cluster a partition.

```
CLUSTER [ VERBOSE ] table_name PARTITION ( partition_name ) [ USING index_name ];
```



- Recluster a table.

```
CLUSTER [ VERBOSE ];
```



Parameters

- **VERBOSE**

  (Optional) Enables the display of progress messages.

- **table_name**

  Specifies the table name.

  Value range: an existing table name

- **[USING index_name]**

  Specifies the index name.

  Value range: an existing index name

  You must specify **index_name** when performing clustering on the table for the first time. If you do not specify **index_name** next time, the table will be clustered based on existing records.

- **partition_name**

  Specifies the partition name.

  Value range: an existing partition name

Examples

- Cluster the table.

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE test_c1(id int, name varchar(20));
gaussdb=# CREATE INDEX idx_test_c1_id ON test_c1(id);
gaussdb=# INSERT INTO test_c1 VALUES (3,'Joe'),(1,'Jack'),(2,'Scott');

-- Query.
gaussdb=# SELECT * FROM test_c1;
 id | name
----+-------
  3 | Joe
  1 | Jack
  2 | Scott
(3 rows)

-- Perform clustering.
```



```
gaussdb=# CLUSTER test_c1 USING idx_test_c1_id;

-- Query.
gaussdb=# SELECT * FROM test_c1;
 id | name
----+-------
  1 | Jack
  2 | Scott
  3 | Joe
(3 rows)

-- Delete.
gaussdb=# DROP TABLE test_c1;
```

- Recluster a table.

```
-- Create a table.
gaussdb=# CREATE TABLE test(col1 int,CONSTRAINT pk_test PRIMARY KEY (col1));

-- An error is reported when the keyword USING is not contained in the first clustering.
gaussdb=# CLUSTER test;
ERROR:  there is no previously clustered index for table "test"

-- Perform clustering.
gaussdb=# CLUSTER test USING pk_test;

-- Insert data.
gaussdb=# INSERT INTO test VALUES (1),(99),(10),(8);

-- Recluster a table.
gaussdb=# CLUSTER VERBOSE test;
INFO:  clustering "public.test" using index scan on "pk_test"(dn_6001 pid=3672)
INFO:  "test": found 0 removable, 4 nonremovable row versions in 1 pages(dn_6001 pid=3672)
DETAIL:  0 dead row versions cannot be removed yet.
CPU 0.00s/0.00u sec elapsed 0.01 sec.
CLUSTER

-- Delete.
gaussdb=# DROP TABLE test;
```

- Cluster a partition.

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE test_c2(id int, info varchar(4)) PARTITION BY RANGE (id)(
    PARTITION p1 VALUES LESS THAN (11),
    PARTITION p2 VALUES LESS THAN (21)
);
gaussdb=# CREATE INDEX idx_test_c2_id1 ON test_c2(id);
gaussdb=# INSERT INTO test_c2 VALUES (6,'ABBB'),(2,'ABAB'),(9,'AAAA');
gaussdb=# INSERT INTO test_c2 VALUES (11,'AAAB'),(19,'BBBA'),(16,'BABA');

-- Query.
gaussdb=# SELECT * FROM test_c2;
 id | info
----+------
  6 | ABBB
  2 | ABAB
  9 | AAAA
 11 | AAAB
 19 | BBBA
 16 | BABA
(6 rows)

-- Perform clustering on partition p2.
gaussdb=# CLUSTER test_c2 PARTITION (p2) USING idx_test_c2_id1;

-- Query.
gaussdb=# SELECT * FROM test_c2;
 id | info
----+------
  6 | ABBB
  2 | ABAB
```



```
 9 | AAAA
 11 | AAAB
 16 | BABA
 19 | BBBA
(6 rows)

-- Delete.
gaussdb=# DROP TABLE test_c2;
```

7.12.8.6 COMMENT

Description

Defines or changes the comment of an object.

Precautions

- Each object stores only one comment. Therefore, you need to modify a comment and issue a new **COMMENT** command to the same object. To delete the comment, write **NULL** at the position of the text string. When an object is deleted, the comment is automatically deleted.
- Currently, there is no security protection for viewing comments. Any user connected to a database can view all the comments for objects in the database. For shared objects such as databases, roles, and tablespaces, comments are stored globally so any user connected to any database in the cluster can see all the comments for shared objects. Therefore, do not put security-critical information in comments.
- To comment objects, you must be an object owner or user granted the COMMENT permission. By default, system administrators have the permission.
- Roles do not have owners, so the rule for **COMMENT ON ROLE** is that you must be an administrator to comment on an administrator role, or have the CREATE ROLE permission to comment on non-administrator roles. A system administrator can comment on all objects.

Syntax

```
COMMENT ON
{
  AGGREGATE agg_name (agg_type [, ...] ) |
  CAST (source_type AS target_type) |
  COLLATION object_name |
  COLUMN { table_name.column_name | view_name.column_name } |
  CONSTRAINT constraint_name ON table_name |
  CONVERSION object_name |
  DATABASE object_name |
  DOMAIN object_name |
  EXTENSION object_name |
  FOREIGN DATA WRAPPER object_name |

  FUNCTION function_name ( [ {[ argname ] [ argmode ] argtype} [, ...] ] ) |
  INDEX object_name |
  LARGE OBJECT large_object_oid |
  OPERATOR operator_name (left_type, right_type) |
  OPERATOR CLASS object_name USING index_method |
  OPERATOR FAMILY object_name USING index_method |
  [ PROCEDURAL ] LANGUAGE object_name |
  ROLE object_name |
  RULE rule_name ON table_name |
  SCHEMA object_name |
  SERVER object_name |
```



```
  TABLE object_name |
  TABLESPACE object_name |
  TEXT SEARCH CONFIGURATION object_name |
  TEXT SEARCH DICTIONARY object_name |
  TEXT SEARCH PARSER object_name |
  TEXT SEARCH TEMPLATE object_name |
  TYPE object_name |
  VIEW object_name |
  TRIGGER trigger_name ON table_name
}
  IS 'text';
```

Parameters

- **agg_name**

  Specifies the new name of an aggregate function.

- **agg_type**

  Specifies the data type of the aggregate function parameters.

- **source_type**

  Specifies the source data type of the cast.

- **target_type**

  Specifies the target data type of the cast.

- **object_name**

  Specifies the name of an object.

- **table_name.column_name**

  view_name.column_name

  Specifies a column name. You can add the table name or view name as the prefix.

- **constraint_name**

  Specifies the name of a table constraint.

- **table_name**

  Specifies the name of a table.

- **function_name**

  Specifies a function name.

- **argname,argmode,argtype**

  Specifies the name, schema, and type of the function parameters.

- **large_object_oid**

  Specifies an OID of a large object.

- **operator_name**

  Specifies the name of the operator.

- **left_type,right_type**

  Specifies the data type of the operator parameters (optionally schema-qualified). If the prefix or suffix operator does not exist, the **NONE** option can be added.

- **trigger_name**

  Specifies the trigger name.

- **text**

  Specifies the comment content.

Examples

```
-- Create a table.
gaussdb=# CREATE TABLE emp(
    empno varchar(7),
    ename varchar(50),
    job varchar(50),
    mgr varchar(7),
    deptno int
);

-- Add comments to a table.
gaussdb=# COMMENT ON TABLE emp IS 'Department table';

-- Add comments to columns.
gaussdb=# COMMENT ON COLUMN emp.empno  IS  'Employee ID';
gaussdb=# COMMENT ON COLUMN emp.ename  IS  'Employee name';
gaussdb=# COMMENT ON COLUMN emp.job    IS  'Job';
gaussdb=# COMMENT ON COLUMN emp.mgr    IS  'Manager ID';
gaussdb=# COMMENT ON COLUMN emp.deptno IS  'Department ID';

-- View table comments.
gaussdb=# \d+
                                        List of relations
 Schema | Name | Type  | Owner |  Size   |                          Storage                           | Description
--------+------+-------+-------+---------+------------------------------------------------------------+-------------
public | emp | table | omm | 0 bytes | {orientation=row,compression=no,storage_type=USTORE,segment=off}
| Department table
(1 row)

-- View column comments.
gaussdb=# \d+ emp
                            Table "public.emp"
 Column |          Type          | Modifiers | Storage  | Stats target | Description
--------+------------------------+-----------+----------+--------------+-------------
 empno  | character varying(7)   |           | extended |              | Employee ID
 ename  | character varying(50)  |           | extended |              | Employee name
 job    | character varying(50)  |           | extended |              | Job
 mgr    | character varying(7)   |           | extended |              | Manager ID
 deptno | integer                |           | plain    |              | Department ID
Has OIDs: no
Options: orientation=row, compression=no, storage_type=USTORE, segment=off


-- Delete the emp table.
gaussdb=# DROP TABLE emp;
```

7.12.8.7 COMMIT | END

Description

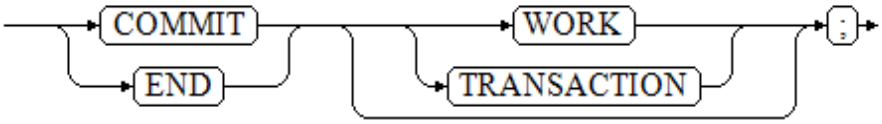
COMMIT or END commits all operations of a transaction.

Precautions

Only the creator of a transaction or the system administrator can run the **COMMIT** command. The creation and commit operations do not need to be in different sessions.

Syntax

```
{ COMMIT | END } [ WORK | TRANSACTION ];
```



Parameters

- **COMMIT | END**

  Commits the current transaction and makes all changes made by the transaction become visible to others.

- **WORK | TRANSACTION**

  Specifies an optional keyword, which has no effect except improving readability.

Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create a table.
gaussdb=# CREATE TABLE tpcds.customer_demographics_t2
(
   CD_DEMO_SK               INTEGER               NOT NULL,
   CD_GENDER                CHAR(1)                     ,
   CD_MARITAL_STATUS        CHAR(1)                     ,
   CD_EDUCATION_STATUS      CHAR(20)                    ,
   CD_PURCHASE_ESTIMATE     INTEGER                     ,
   CD_CREDIT_RATING         CHAR(10)                    ,
   CD_DEP_COUNT             INTEGER                     ,
   CD_DEP_EMPLOYED_COUNT    INTEGER                     ,
   CD_DEP_COLLEGE_COUNT     INTEGER
)
;

-- Start a transaction.
gaussdb=# START TRANSACTION;

-- Insert data.
gaussdb=# INSERT INTO tpcds.customer_demographics_t2 VALUES(1,'M', 'U', 'DOCTOR DEGREE', 1200,
'GOOD', 1, 0, 0);
gaussdb=# INSERT INTO tpcds.customer_demographics_t2 VALUES(2,'F', 'U', 'MASTER DEGREE', 300, 'BAD',
1, 0, 0);

-- Commit the transaction to make all changes permanent.
gaussdb=# COMMIT;

-- Query data.
gaussdb=# SELECT * FROM tpcds.customer_demographics_t2;

-- Delete the tpcds.customer_demographics_t2 table.
gaussdb=# DROP TABLE tpcds.customer_demographics_t2;

-- Delete the schema.
gaussdb=# DROP SCHEMA tpcds;
```

Helpful Links

ROLLBACK

7.12.8.8 COMMIT PREPARED

Description

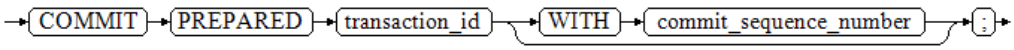
Commits a prepared two-phase transaction.

Precautions

- The function is only available in maintenance mode (when the GUC parameter **xc_maintenance_mode** is **on**). Exercise caution when enabling the mode. It is used by maintenance engineers for troubleshooting. Common users should not use the mode.
- Only the creator of a transaction or a system administrator can run this command. The creation and commit operations do not need to be in different sessions.
- The transaction function is maintained automatically by the database, and should be not visible to users.

Syntax

```
COMMIT PREPARED transaction_id [ WITH commit_sequence_number ];
```



Parameters

- **transaction_id**

  Specifies the identifier of the transaction to be committed. The identifier must be different from those for current prepared transactions.

- **commit_sequence_number**

  Specifies the sequence number of the transaction to be committed. It is a 64-bit, incremental, unsigned number.

Example

```
-- Start.
gaussdb=# BEGIN;

-- Prepare a transaction whose identifier is trans_test.
gaussdb=# PREPARE TRANSACTION 'trans_test';

-- Create a table.
gaussdb=# CREATE TABLE item1(id int);

--Commit the transaction whose identifier is trans_test.
gaussdb=# COMMIT PREPARED 'trans_test';

-- Drop the table.
gaussdb=# DROP TABLE item1;
```

Helpful Links

PREPARE TRANSACTION and **ROLLBACK PREPARED**

7.12.8.9 COPY

Description

Copies data between tables and files.

COPY FROM copies data from a file to a table, and COPY TO copies data from a table to a file.

Precautions

- When the **enable_copy_server_files** parameter is disabled, only the initial user is allowed to run the **COPY FROM FILENAME** or **COPY TO FILENAME** statement. When the **enable_copy_server_files** parameter is enabled, users with the **SYSADMIN** permission or users who inherit the built-in role permission **gs_role_copy_files** are allowed to run the **COPY FROM FILENAME** or **COPY TO FILENAME** statement. By default, database configuration files and key files are not allowed, and you can run **COPY FROM FILENAME** or **COPY TO FILENAME** for certificate files and audit logs to prevent unauthorized users from viewing or modifying sensitive files. When **enable_copy_server_files** is enabled, the administrator can use the GUC parameter **safe_data_path** to set the path for common users to import and export to the subpath of the set path. If this GUC parameter is not set (by default), the path used by common users is not blocked. This parameter reports an error for **...** in the path of the COPY statement.
- COPY applies only to tables but not views.
- COPY TO requires the SELECT permission on the table to be read, and COPY FROM requires the INSERT permission on the table to be inserted.
- If a list of columns is specified, COPY copies only the data of the specified columns between the file and the table. If a table has any columns that are not in the column list, **COPY FROM** inserts default values for those columns.
- If a data source file is specified, the server must be able to access the file. If **STDIN** is specified, data flows between the client and the server. When entering data, use the **TAB** key to separate the columns of the table and use a backslash and a period (\.) in a new row to indicate the end of the input.
- COPY FROM throws an error if any row in the data file contains more or fewer columns than expected.
- The end of the data can be represented by a line that contains only backslashes and periods (\.). If data is read from a file, the end flag is unnecessary. If data is copied between client applications, an end tag must be provided.
- In COPY FROM, **\N** is an empty string. To enter the actual value **\N**, use **\\N**.
- COPY FROM can preprocess data using column expressions, but column expressions do not support subqueries.
- When a data format error occurs during COPY FROM execution, the transaction is rolled back. However, the error information is insufficient, making it difficult to locate the error data from a large amount of raw data.
- COPY FROM and COPY TO apply to low concurrency and local import and export of a small amount of data.

- If the target table has triggers, COPY is supported.
- Ensure that the generated column is not in the list of the specified column in the COPY statement. When COPY... TO is used to export data, if no column list is specified, all columns except the generated columns in the table are exported. When COPY... FROM is used to import data, the generated columns are automatically updated and saved as common columns.
- COPY is a server command and its operating environment is the same as that of the database server process. \COPY is a client meta-command and its operating environment is the same as that of gsql on the client. Note that when the database and gsql are used in the cloud environment, the COPY and \COPY statements both use the paths in the cloud environment. When the database is used in the cloud environment and gsql is used outside the cloud environment, the COPY statement uses the path inside the cloud environment, and the \COPY statement uses the path outside the cloud environment.
- During the export using COPY TO, if the column data in the table contains the **'\0'** character, the column data will be truncated during the export. Only the data before **'\0'** will be exported.
- The import and export statistics and error information are stored in the gs_copy_summary and pgxc_copy_error_log system catalogs, respectively. Objects that may cause privilege escalation, such as RULE, TRIGGER, index functions, row-level security, CHECK constraints, **GENERATED** columns, **DEFAULT** columns, and **ON UPDATE** columns, cannot be contained in the system catalogs. Otherwise, the system considers that the objects are created by malicious users, reports an error, and exits.

Syntax

- Copy data from a file to a table.

```
COPY table_name [ ( column_name [, ...] ) ]
   FROM { 'filename' | STDIN }
   [ [ USING ] DELIMITERS 'delimiters' ]
   [ WITHOUT ESCAPING ]
   [ LOG ERRORS ]
   [ LOG ERRORS DATA ]
   [ REJECT LIMIT 'limit' ]
   [ [ WITH ]
       ( option [, ...] )
       | ( copy_option [, ...] )
       | [ TRANSFORM ( { column_name [ data_type ] [ AS transform_expr ] } [, ...] ) ]
       | [ FIXED FORMATTER ( { column_name( offset, length ) } [, ...] ) ]
   ]
```

  NOTE

  The **fixed formatter** syntax is compatible with **copy_option** but incompatible with **option**. **copy_option** is incompatible with **option**. **transform** is compatible with **copy_option** and **fixed formatter**.

- Copy data from a table to a file.

```
COPY table_name [ ( column_name [, ...] ) ]
   TO { 'filename' | STDOUT }
   [ [ USING ] DELIMITERS 'delimiters' ]
   [ WITHOUT ESCAPING ]
   [ [ WITH ]
       ( option [, ...] )
       | ( copy_option[, ...] )
       | [ FIXED FORMATTER ( { column_name( offset, length ) } [, ...] ) ]
   ]
```



```
COPY query
   TO { 'filename' | STDOUT }
   [ WITHOUT ESCAPING ]
   [ [ WITH ]
       ( option [, ...] )
       | ( copy_option [, ...] )
       | [ FIXED FORMATTER ( { column_name( offset, length ) } [, ...] ) ]
   ]
```

NOTE

- The syntax constraints of COPY TO are as follows:

  (query) is incompatible with [USING] DELIMITERS. If the data comes from a query result, COPY TO cannot specify [USING] DELIMITERS.

- Use spaces to separate **copy_option** following FIXED FORMATTTER.
- **copy_option** is the native parameter, while **option** is the parameter imported by a compatible foreign table.

The syntax of the optional parameter **option** is as follows:

```
FORMAT 'format_name'
| FORMAT binary
| OIDS [ boolean ]
| DELIMITER 'delimiter_character'
| NULL 'null_string'
| HEADER [ boolean ]
| USEEOF [ boolean ]
| FILEHEADER 'header_file_string'
| FREEZE [ boolean ]
| QUOTE 'quote_character'
| ESCAPE 'escape_character'
| EOL 'newline_character'
| NOESCAPING [ boolean ]
| FORCE_QUOTE { ( column_name [, ...] ) | * }
| FORCE_NOT_NULL ( column_name [, ...] )
| ENCODING 'encoding_name'
| IGNORE_EXTRA_DATA [ boolean ]
| FILL_MISSING_FIELDS [ boolean ]
| COMPATIBLE_ILLEGAL_CHARS [ boolean ]
| DATE_FORMAT 'date_format_string'
| TIME_FORMAT 'time_format_string'
| TIMESTAMP_FORMAT 'timestamp_format_string'
| SMALLDATETIME_FORMAT 'smalldatetime_format_string'
```

The syntax of the optional parameter **copy_option** is as follows:

```
OIDS
| NULL 'null_string'
| HEADER
| USEEOF
| FILEHEADER 'header_file_string'
| FREEZE
| FORCE_NOT_NULL column_name [, ...]
| FORCE_QUOTE { column_name [, ...] | * }
| BINARY
| CSV
| QUOTE [ AS ] 'quote_character'
| ESCAPE [ AS ] 'escape_character'
| EOL 'newline_character'
| ENCODING 'encoding_name'
| IGNORE_EXTRA_DATA
| FILL_MISSING_FIELDS [ { 'one' | 'multi' } ]
| COMPATIBLE_ILLEGAL_CHARS
| DATE_FORMAT 'date_format_string'
| TIME_FORMAT 'time_format_string'
| TIMESTAMP_FORMAT 'timestamp_format_string'
| SMALLDATETIME_FORMAT 'smalldatetime_format_string'
| SKIP int_number
| WHEN { ( start - end ) | column_name } { = | != } 'string'
```



```
| SEQUENCE ( { column_name ( integer [, incr] ) [, ...] } )
| FILLER ( { column_name [, ...] } )
| CONSTANT ( { column_name 'constant_string' [, ...] } )
```

Parameters

- **query**

  Specifies that the results will be copied.

  Valid value: a **SELECT** or **VALUES** command in parentheses

- **table_name**

  Specifies the name (possibly schema-qualified) of an existing table.

  Value range: an existing table name

- **column_name**

  Specifies an optional list of columns to be copied.

  Value range: any columns. All columns will be copied if no column list is specified.

- **STDIN**

  Specifies that input comes from the standard input. In the input, table columns are separated by tabs and each row ends with a backslash and a period (\.).

- **STDOUT**

  Specifies that output goes to the standard output.

- **FIXED**

  Fixes column length. When the column length is fixed, **DELIMITER**, **NULL**, and **CSV** cannot be specified. When **FIXED** is specified, **BINARY**, **CSV**, and **TEXT** cannot be specified by **option** or **copy_option**.

  NOTE

  The definition of fixed length is as follows:

  - The column length of each record is the same.
  - Spaces are used for column padding. Columns of the numeric type are left-aligned and columns of the string type are right-aligned.
  - No delimiters are used between columns.

- **[USING] DELIMITERS 'delimiters'**

  String that separates columns within each row (line) of the file. It cannot be larger than 10 bytes.

  Value range: The delimiter in text format cannot include any of the following characters: \.abcdefghijklmnopqrstuvwxyz0123456789, but has no restriction for the CSV format.

  Value range: The default value is a tab character in text format and a comma in CSV format.

  NOTE

  Both DELIMITER and DELIMITERS can specify delimiters. DELIMITERS can be followed by brackets, but DELIMITER cannot be directly followed by brackets. Otherwise, a syntax error is reported.

- **WITHOUT ESCAPING**

Specifies, in the TEXT format, whether to escape the backslash (\) and its following characters.

Value range: text only

- **LOG ERRORS**

  If this parameter is specified, the error tolerance mechanism for data type errors in the COPY FROM statement is enabled.

  Value range: a value set while data is imported using COPY FROM.

  NOTE

  The restrictions of this error tolerance parameter are as follows:

  - This error tolerance mechanism captures only the data type errors (DATA_EXCEPTION) that occur during data parsing of COPY FROM on the primary node of the database.
  - If existing error tolerance parameters (for example, **IGNORE_EXTRA_DATA**) of the COPY statement are enabled, the error of the corresponding type will be processed as specified by the parameters and no error will be reported. Therefore, the error table does not contain such error data.

- **LOG ERRORS DATA**

  The differences between LOG ERRORS DATA and LOG ERRORS are as follows:

  a. LOG ERRORS DATA fills the **rawrecord** column in the error tolerance table.

  b. Only users with the super permission can use the parameter options of **LOG ERRORS DATA**.

  CAUTION

  - If error content is too complex, it may fail to be written to the error tolerance table by using **LOG ERRORS DATA**, causing the task failure.
  - For errors that cannot be read in certain code, the error codes are ERRCODE_CHARACTER_NOT_IN_REPERTOIRE and ERRCODE_UNTRANSLATABLE_CHARACTER. The rawrecord column is not recorded.

- **REJECT LIMIT 'limit'**

  Used with the **LOG ERRORS** option to set the upper limit of the tolerated errors in the COPY FROM statement. If the number of errors exceeds the limit, later errors will be reported based on the original mechanism.

  Value range: a positive integer (1 to *INTMAX*) or **unlimited**

  Default value: If **LOG ERRORS** is not specified, an error will be reported. If LOG ERRORS is specified, the default value is **0**.

  NOTE

  In the error tolerance mechanism described in the description of **LOG ERRORS**, the count of REJECT LIMIT is calculated based on the number of data parsing errors on the primary node of the database where the COPY FROM statement is executed, not based on the number of all errors on the primary node.

- **FORMATTER**

Defines the place of each column in the data file in fixed length mode. Defines the place of each column in the data file in the **column(***offset*,*length***)** format.

Value range:

- The value of **offset** must be larger than 0. The unit is byte.
- The value of **length** must be larger than 0. The unit is byte.

The total length of all columns must be less than 1 GB.

Replace columns that are not in the file with null.

- **OPTION { option_name ' value ' }**

  Specifies all types of parameters of a compatible foreign table.

  - FORMAT

    Specifies the format of the source data file in the foreign table.

    Value range: **CSV**, **TEXT**, **FIXED**, and **BINARY**

    - The CSV file can process newline characters efficiently, but cannot process certain special characters well.
    - The TEXT file can process certain special characters efficiently, but cannot process newline characters well.
    - In FIXED files, the column length of each record is the same. Spaces are used for padding, and the excessive part will be truncated.
    - All data in the BINARY file is stored/read as binary format rather than as text. It is faster than the text and CSV formats, but a binary-format file is less portable.

    Default value: **TEXT**

  - DELIMITER

    Specifies the character that separates columns within each row (line) of the file.

    NOTE

    - The value of **delimiter** cannot be **\r** or **\n**.
    - A delimiter cannot be the same as the null value. The delimiter for the CSV format cannot be same as the **quote** value.
    - The delimiter for the TEXT format data cannot contain lowercase letters, digits, or special characters (.\).
    - The data length of a single row should be less than 1 GB. A row that has many columns using long delimiters cannot contain much valid data.
    - You are advised to use multi-character delimiters or invisible delimiters. For example, you can use multi-characters (such as $^&) and invisible characters (such as 0x07, 0x08, and 0x1b).

    Value range: a multi-character delimiter within 10 bytes

    Default value:

    - A tab character in text format
    - A comma (,) in CSV format

- No delimiter in FIXED format

– NULL

  Specifies the string that represents a null value.

  Value range:

  - A null value cannot be **\r** or **\n**. The maximum length is 100 characters.
  - A null value cannot be the same as the **delimiter** or **quote** value.

  Default value:

  - The default value for the CSV format is an empty string without quotation marks.
  - The default value for the TEXT format is **\N**.

– HEADER

  Specifies whether a file contains a header with the names of each column in the file. **header** is available only for CSV and FIXED files.

  When data is imported, if **header** is **on**, the first row of the data file will be identified as the header and ignored. If **header** is **off**, the first row will be identified as a data row.

  When data is exported, if header is **on**, **fileheader** must be specified. If **header** is **off**, an exported file does not contain a header.

  Value range: **true**/**on** or **false**/**off**.

  Default value: **false**

– USEEOF

  The system does not report an error for "\." in the imported data.

  Value range: **true**/**on** or **false**/**off**.

  Default value: **false**

– QUOTE

  Specifies a quoted character string for a CSV file.

  Default value: double quotation marks ("")

  NOTE

  - The value of **quote** cannot be the same as that of the **delimiter** or **null** parameter.
  - The value of **quote** must be a single-byte character.
  - You are advised to set **quote** to an invisible character, such as **0x07**, **0x08**, or **0x1b**.

– ESCAPE

  Specifies an escape character for a CSV file. The value must be a single-byte character.

  Default value: double quotation marks (""). If the value is the same as that of **quote**, it will be replaced by **'\0'**.

– EOL 'newline_character'

  Specifies the newline character style of the imported or exported data file.

Value range: multi-character newline characters within 10 bytes. Common newline characters include **\r** (0x0D), **\n** (0x0A), and **\r\n** (0x0D0A). Special newline characters include **$** and **#**.

NOTE

- The **EOL** parameter supports only the TEXT format for data import and export and does not support the CSV or FIXED format for data import. For forward compatibility, the EOL parameter can be set to **0x0D** or **0x0D0A** for data export in the CSV or FIXED format.
- The value of **EOL** cannot be the same as that of the **delimiter** or **null** parameter.
- The EOL parameter value cannot contain the following characters: .abcdefghijklmnopqrstuvwxyz0123456789.

– FORCE_QUOTE { (column_name [, ...]) | * }

In **CSV COPY TO** mode, forces quotation marks to be used for all non-null values in each specified column. The asterisk (*) indicates all columns. Null values are not quoted.

Value range: an existing column name

– FORCE_NOT_NULL (column_name [, ...])

Assigns a value to a specified column in **CSV COPY FROM** mode.

Value range: an existing column name

– ENCODING

Specifies the encoding format of a data file. The default value is the current client encoding format.

– IGNORE_EXTRA_DATA

Specifies whether to ignore excessive columns when the number of data source files exceeds the number of foreign table columns. This parameter is used only during data import.

Value range: **true**/**on** or **false**/**off**

- **true**/**on**: If the number of columns in a data source file is greater than that defined by the foreign table, the extra columns at the end of a row are ignored.
- **false**/**off**: If the number of columns in a data source file is greater than that defined by the foreign table, the following error message is reported:
  ```
  extra data after last expected column
  ```

Default value: **false**

NOTICE

If a newline character at the end of a row is missing and the row and another row are integrated into one, data in another row is ignored after the parameter is set to **true**.

– COMPATIBLE_ILLEGAL_CHARS

Specifies whether to tolerate invalid characters during data import. The parameter is valid only for data import using **COPY FROM**.

Value range: **true**/**on** or **false**/**off**.

- **true**/**on**: No error message is reported and data import is not interrupted when there are invalid characters. Invalid characters are converted into valid ones, and then imported to the database.
- **false**/**off**: An error occurs when there are invalid characters, and the import stops.

Default value: **false** or **off**

NOTE

The rules for converting invalid characters are as follows:

1. **\0** is converted to a space.

2. Other invalid characters are converted to question marks.

3. When **compatible_illegal_chars** is set to **true**/**on**, after invalid characters such as **NULL**, **DELIMITER**, **QUOTE**, and **ESCAPE** are converted to spaces or question marks, an error message stating "illegal chars conversion may confuse COPY escape 0x20" will be displayed to remind you of possible parameter confusion caused by the conversion.

4. When the GUC parameter **copy_special_character_version** is set to **'no_error'**, **compatible_illegal_chars** cannot be set to **true** or **on**.

5. When the GUC parameter **copy_special_character_version** is set to **'no_error'**, invalid characters will not be checked during the import and will be displayed as garbled characters in query results. Exercise caution when enabling this parameter. You can use the **LOG ERRORS** or **LOG ERRORS DATA** parameter in the COPY statement to code errors to an error table.

– FILL_MISSING_FIELDS

Specifies how to handle the problem that the last column of a row in a source data file is lost during data import.

Value range: **true**/**on** or **false**/**off**.

Default value: **false** or **off**

– DATE_FORMAT

Specifies the DATE format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using COPY FROM.

Value range: a valid DATE value For details, see **Date and Time Processing Functions and Operators**.

NOTE

You can use the **TIMESTAMP_FORMAT** parameter to set the DATE format to **TIMESTAMP** for data import. For details, see **TIMESTAMP_FORMAT** below.

– TIME_FORMAT

Specifies the TIME format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using COPY FROM.

Value range: a valid TIME value. Time zones cannot be used. For details, see **Date and Time Processing Functions and Operators**.

– TIMESTAMP_FORMAT

Specifies the TIMESTAMP format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using COPY FROM.

Value range: a valid TIMESTAMP value. Time zones cannot be used. For details, see **Date and Time Processing Functions and Operators**.

- SMALLDATETIME_FORMAT

  Specifies the SMALLDATETIME format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using COPY FROM.

  Value range: a valid SMALLDATETIME value. For details, see **Date and Time Processing Functions and Operators**.

- **COPY_OPTION { option_name ' value ' }**

  Specifies all types of native parameters of COPY.

  - NULL null_string

    Specifies the string that represents a null value.

    NOTICE

    When using COPY FROM, any data item that matches this string will be stored as a null value, so make sure that you use the same string as you used with COPY TO.

    Value range:

    - A null value cannot be **\r** or **\n**. The maximum length is 100 characters.
    - A null value cannot be the same as the **delimiter** or **quote** value.

    Default value:

    - The default value for the TEXT format is **\N**.
    - The default value for the CSV format is an empty string without quotation marks.

  - HEADER

    Specifies whether a file contains a header with the names of each column in the file. **header** is available only for CSV and FIXED files.

    When data is imported, if **header** is **on**, the first row of the data file will be identified as the header and ignored. If **header** is **off**, the first row will be identified as a data row.

    When data is exported, if header is **on**, **fileheader** must be specified. If **header** is **off**, an exported file does not contain a header.

  - USEEOF

    The system does not report an error for "\." in the imported data.

  - FILEHEADER

Specifies a file that defines the content in the header for exported data. The file contains data description of each column.

NOTICE

- This parameter is available only when **header** is **on** or **true**.
- **fileheader** specifies an absolute path.
- The file can contain only one row of header information, and ends with a newline character. Excess rows will be discarded. (Header information cannot contain newline characters.)
- The length of the file including the newline character cannot exceed 1 MB.

– FREEZE

Sets the COPY loaded data row as frozen, like these data rows have executed VACUUM FREEZE.

This is a performance option of initial data loading. The data will be frozen only when the following three requirements are met:

- The table being loaded has been created or truncated in the same transaction before copying.
- There are no cursors open in the current transaction.
- There are no original snapshots in the current transaction.

NOTE

When **COPY** is completed, all the other sessions will see the data immediately. However, this violates the general principle of MVCC visibility, and users should understand that this may cause potential risks.

– FORCE NOT NULL column_name [, ...]

Assigns a value to a specified column in **CSV COPY FROM** mode. If the column is null, its value is regarded as a string of 0 characters.

Value range: an existing column name

– FORCE QUOTE { column_name [, ...] | * }

In **CSV COPY TO** mode, forces quotation marks to be used for all non-null values in each specified column. The asterisk (*) indicates all columns. Null values are not quoted.

Value range: an existing column name

– BINARY

Specifies that data is stored and read in binary mode instead of text mode.

NOTE

- In binary mode, you cannot declare **DELIMITER**, **NULL**, or **CSV**.
- When **BINARY** is specified, **CSV**, **FIXED**, and **TEXT** cannot be specified through **option** or **copy_option**.
- If the GUC parameter **copy_special_character_version** is set to **'no_error'**, invalid characters will not be checked during the import and will be displayed as garbled characters in query results. The database server code must be the same as the file code. Exercise caution when enabling this parameter. You can use the **LOG ERRORS** or **LOG ERRORS DATA** parameter in the COPY statement to code errors to an error table.
- In binary mode, **copy_special_character_version** is set to **'no_error'**, and it takes effect only for columns of the TEXT, CHAR, VARCHAR, NVARCHAR2, and CLOB types.

– CSV

Enables the CSV mode. When **CSV** is specified, **BINARY**, **FIXED**, and **TEXT** cannot be specified through **option** or **copy_option**.

– QUOTE [AS] 'quote_character'

Specifies a quoted character string for a CSV file.

Default value: double quotation marks ("").

NOTE

- The value of **quote** cannot be the same as that of the **delimiter** or **null** parameter.
- The value of **quote** must be a single-byte character.
- You are advised to set **quote** to an invisible character, such as **0x07**, **0x08**, or **0x1b**.

– ESCAPE [AS] 'escape_character'

Specifies an escape character for a CSV file. The value must be a single-byte character.

Default value: double quotation marks (""). If the value is the same as that of **quote**, it will be replaced by **'\0'**.

– EOL 'newline_character'

Specifies the newline character style of the imported or exported data file.

Value range: multi-character newline characters within 10 bytes. Common newline characters include **\r** (0x0D), **\n** (0x0A), and **\r\n** (0x0D0A). Special newline characters include **$** and **#**.

NOTE

- The **EOL** parameter supports only the TEXT format for data import and export and does not support the CSV or FIXED format. For forward compatibility, the **EOL** parameter can be set to **0x0D** or **0x0D0A** for data export in the CSV or FIXED format.
- The value of **EOL** cannot be the same as that of the **delimiter** or **null** parameter.
- The EOL parameter value cannot contain the following characters: .abcdefghijklmnopqrstuvwxyz0123456789.

– ENCODING 'encoding_name'

Specifies the name of a file encoding format.

Value range: a valid encoding format

Default value: current encoding format

- IGNORE_EXTRA_DATA

  Specifies that when the number of data source files exceeds the number of foreign table columns, excess columns at the end of the row are ignored. This parameter is used only during data import.

  If this parameter is not used and the number of columns in the data source file is greater than that defined in the foreign table, the following error information is displayed:

  ```
  extra data after last expected column
  ```

- COMPATIBLE_ILLEGAL_CHARS

  Specifies that invalid characters are tolerated during data import. Invalid characters are converted and then imported to the database. No error is reported and the import is not interrupted. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using COPY FROM.

  If this parameter is not used, an error is reported when invalid characters are encountered during the import, and the import is interrupted.

  NOTE

  The rules for converting invalid characters are as follows:

  1. **\0** is converted to a space.

  2. Other invalid characters are converted to question marks.

  3. When **compatible_illegal_chars** is set to **true**/**on**, after invalid characters such as **NULL**, **DELIMITER**, **QUOTE**, and **ESCAPE** are converted to spaces or question marks, an error message stating "illegal chars conversion may confuse COPY escape 0x20" will be displayed to remind you of possible parameter confusion caused by the conversion.

  4. When the GUC parameter **copy_special_character_version** is set to **'no_error'**, **compatible_illegal_chars** cannot be set to **true** or **on**.

  5. When the GUC parameter **copy_special_character_version** is set to **'no_error'**, invalid characters will not be checked during the import and will be displayed as garbled characters in query results. Exercise caution when enabling this parameter. You can use the **LOG ERRORS** or **LOG ERRORS DATA** parameter in the COPY statement to code errors to an error table.

- FILL_MISSING_FIELDS [{ 'one' | 'multi' }]

  Specifies how to handle the problem that the last column of a row in a source data file is lost during data import. If **one** or **multi** is not specified or **one** is specified, the missing of the last column is handled in the default mode. If **multi** is specified, the missing of the last multiple columns are handled in the default mode.

  Value range: **true**/**on** or **false**/**off**.

  Default value: **false** or **off**

NOTICE

Do not specify this option. Currently, it does not enable error tolerance, but will make the parser ignore the said errors during data parsing on the primary node of the database. Such errors will not be recorded in the COPY error table (enabled using **LOG ERRORS REJECT LIMIT**) but will be reported later by database node. Therefore, do not specify this option.

- DATE_FORMAT 'date_format_string'

  Specifies the DATE format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using COPY FROM.

  Value range: a valid DATE value For details, see **Date and Time Processing Functions and Operators**.

  NOTE

  You can use the **TIMESTAMP_FORMAT** parameter to set the DATE format to **TIMESTAMP** for data import. For details, see **TIMESTAMP_FORMAT** below.

- TIME_FORMAT 'time_format_string'

  Specifies the TIME format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using COPY FROM.

  Value range: a valid TIME value. Time zones cannot be used. For details, see **Date and Time Processing Functions and Operators**.

- TIMESTAMP_FORMAT 'timestamp_format_string'

  Specifies the TIMESTAMP format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using COPY FROM.

  Value range: a valid TIMESTAMP value. Time zones cannot be used. For details, see **Date and Time Processing Functions and Operators**.

- SMALLDATETIME_FORMAT 'smalldatetime_format_string'

  Specifies the SMALLDATETIME format for data import. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. The parameter is valid only for data import using COPY FROM.

  Value range: a valid SMALLDATETIME value. For details, see **Date and Time Processing Functions and Operators**.

- TRANSFORM ({ column_name [data_type] [AS transform_expr] } [, ...])

  Specify the conversion expression of each column in the table. **data_type** specifies the data type of the column in the expression parameter. **transform_expr** is the target expression that returns the result value whose data type is the same as that of the target column in the table. For details about the expression, see **Expressions**.

- SKIP int_number

Specifies that the first *int_number* rows of the data file are skipped during data import.

- WHEN { (start - end) | column_name } { = | != } 'string'

  When data is imported, each row of data is checked. Only the rows that meet the WHEN condition are imported to the table.

- SEQUENCE ({ column_name (integer [, incr]) [, ...] })

  During data import, columns modified by SEQUENCE do not read data from the data file. The values are incremented by the value of **incr** based on the specified integer. If **incr** is not specified, the values are incremented from 1 by default.

- FILLER ({ column_name [, ...] })

  When data is imported, the column modified by FILLER is discarded after being read from the data file.

  NOTE

  To use FILLER, you need to specify the list of columns to be copied. Data is processed based on the position of the **filler** column in the column list.

- CONSTANT ({ column_name 'constant_string' [, ...] })

  When data is imported, the column modified by CONSTANT is not read from the data file, and **constant_string** is used to assign a value to the column.

The following special backslash sequences are recognized by COPY FROM:

- **\b**: Backslash (ASCII 8)
- **\f**: Form feed (ASCII 12)
- **\n**: Newline character (ASCII 10)
- **\r**: Carriage return character (ASCII 13)
- **\t**: Tab (ASCII 9)
- **\v**: Vertical tab (ASCII 11)
- **\digits**: Backslash followed by one to three octal digits specifies that the ASCII value is the character with that numeric code.
- **\xdigits**: Backslash followed by an x and one or two hex digits specifies the character with that numeric code.

Permission Control Examples

```
gaussdbs=> copy t1 from '/home/xy/t1.csv';
ERROR:  COPY to or from a file is prohibited for security concerns
HINT:  Anyone can COPY to stdout or from stdin. gsql's \copy command also works for anyone.
gaussdb=> grant gs_role_copy_files to xxx;
```

This error occurs because a non-initial user does not have the COPY permission. To solve this problem, enable the **enable_copy_server_files** parameter. Then, the administrator can use the COPY function, and common users need to join the **gs_role_copy_files** group.

Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create the tpcds.ship_mode table.
```



```
gaussdb=# CREATE TABLE tpcds.ship_mode
(
    SM_SHIP_MODE_SK          INTEGER               NOT NULL,
    SM_SHIP_MODE_ID          CHAR(16)              NOT NULL,
    SM_TYPE                  CHAR(30)                      ,
    SM_CODE                  CHAR(10)                      ,
    SM_CARRIER               CHAR(20)                      ,
    SM_CONTRACT              CHAR(20)
)
;

-- Insert a single data record into the tpcds.ship_mode table.
gaussdb=# INSERT INTO tpcds.ship_mode VALUES (1,'a','b','c','d','e');

-- Copy data from the tpcds.ship_mode file to the /home/omm/ds_ship_mode.dat file.
gaussdb=# COPY tpcds.ship_mode TO '/home/omm/ds_ship_mode.dat';

-- Output tpcds.ship_mode to STDOUT.
gaussdb=# COPY tpcds.ship_mode TO STDOUT;

-- Output the data of tpcds.ship_mode to STDOUT. The parameters are as follows: The separator is ',' (delimiter',') and the encoding format is UTF8 (encoding'utf8').
gaussdb=# COPY tpcds.ship_mode TO STDOUT WITH (delimiter ',', encoding 'utf8');

-- Output the data of tpcds.ship_mode to STDOUT. The parameters are as follows: The import format is CSV (format'CSV'), and the exported content of the SM_SHIP_MODE_SK column is enclosed in quotation marks (force_quote(SM_SHIP_MODE_SK)).
gaussdb=# COPY tpcds.ship_mode TO STDOUT WITH (format 'CSV', force_quote(SM_SHIP_MODE_SK));

-- Create the tpcds.ship_mode_t1 table.
gaussdb=# CREATE TABLE tpcds.ship_mode_t1
(
    SM_SHIP_MODE_SK          INTEGER               NOT NULL,
    SM_SHIP_MODE_ID          CHAR(16)              NOT NULL,
    SM_TYPE                  CHAR(30)                      ,
    SM_CODE                  CHAR(10)                      ,
    SM_CARRIER               CHAR(20)                      ,
    SM_CONTRACT              CHAR(20)
)
;

-- Copy data from STDIN to the tpcds.ship_mode_t1 table.
gaussdb=# COPY tpcds.ship_mode_t1 FROM STDIN;

-- Copy data from the /home/omm/ds_ship_mode.dat file to the tpcds.ship_mode_t1 table.
gaussdb=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat';

-- Copy data from the /home/omm/ds_ship_mode.dat file to the tpcds.ship_mode_t1 table, convert the data using the TRANSFORM expression, and insert the 10 characters on the left of the SM_TYPE column into the table.
gaussdb=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat' TRANSFORM (SM_TYPE AS LEFT(SM_TYPE, 10));

-- Copy data from the /home/omm/ds_ship_mode.dat file to the tpcds.ship_mode_t1 table, with the import format set to TEXT (format 'text'), the delimiter set to '\t' (delimiter E'\t'), excessive columns ignored (ignore_extra_data 'true'), and characters not escaped (noescaping 'true').
gaussdb=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat' WITH(format 'text', delimiter E'\t', ignore_extra_data 'true', noescaping 'true');

-- Copy data from the /home/omm/ds_ship_mode.dat file to the tpcds.ship_mode_t1 table, with the import format set to FIXED, fixed-length format specified (FORMATTER(SM_SHIP_MODE_SK(0, 2), SM_SHIP_MODE_ID(2,16), SM_TYPE(18,30), SM_CODE(50,10), SM_CARRIER(61,20), SM_CONTRACT(82,20))), excessive columns ignored (ignore_extra_data), and headers included (header).
gaussdb=# COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat' FIXED FORMATTER(SM_SHIP_MODE_SK(0, 2), SM_SHIP_MODE_ID(2,16), SM_TYPE(18,30), SM_CODE(50,10), SM_CARRIER(61,20), SM_CONTRACT(82,20)) header ignore_extra_data;

-- Delete tables and the schema.
gaussdb=# DROP TABLE tpcds.ship_mode;
```



```
gaussdb=# DROP TABLE tpcds.ship_mode_t1;
gaussdb=# DROP SCHEMA tpcds;
```

7.12.8.10 CREATE AGGREGATE

Description

Creates an aggregate function.

Syntax

```
CREATE AGGREGATE name ( input_data_type [ , ... ] ) (
    SFUNC = sfunc,
    -- SFUNC1 = sfunc, // Outdated format, equivalent to SFUNC = sfunc.
    STYPE = state_data_type
    -- STYPE1 = state_data_type, // Outdated format, equivalent to STYPE = state_data_type
    [ , FINALFUNC = ffunc ]
    [ , INITCOND = initial_condition ]
    -- [ , INITCOND1 = initial_condition ] // Outdated format, equivalent to INITCOND = initial_condition
    [ , SORTOP = sort_operator ]
    [ , CFUNC = collection_func ]
    [ , INITCOLLECT = initial_collection_condition ]
    [ , IFUNC = init_func ]
    [ , SHIPPABLE = is_shippable ]
);

or the old syntax

CREATE AGGREGATE name (
    BASETYPE = base_type,
    SFUNC = sfunc,
    -- SFUNC1 = sfunc, // Outdated format, equivalent to SFUNC = sfunc.
    STYPE = state_data_type
    -- STYPE1 = state_data_type, // Outdated format, equivalent to STYPE = state_data_type
    [ , FINALFUNC = ffunc ]
    [ , INITCOND = initial_condition ]
    -- [ , INITCOND1 = initial_condition ] // Outdated format, equivalent to INITCOND = initial_condition
    [ , SORTOP = sort_operator ]
    [ , CFUNC = collection_func ]
    [ , INITCOLLECT = initial_collection_condition ]
    [ , IFUNC = init_func ]
);
```

Parameters

- **name**

  Name (optionally schema-qualified) of the aggregate function to be created.

- **input_data_type**

  Data type of the input to be processed by the aggregate function. To create a zero-parameter aggregate function, you can use an asterisk (*) instead of a list of input data types. (count(*) is an instance of this aggregate function.)

- **base_type**

  In the CREATE AGGREGATE syntax, the input data type is specified by the **basetype** parameter instead of following **name**. Note that the previous syntax allows only one input parameter. To create a zero-parameter aggregate function, you can set **basetype** to **ANY** instead of ***.

- **sfunc**

  Name of the state conversion function that will be called on each input line. For an aggregate function with N parameters, **sfunc** must have more than

one parameter. The first parameter is of the **state_data_type** type, and the other parameters match the declared input data types. The function must return a value of the **state_data_type** type. This function accepts the current state value and the current input data, and returns the next state value. The default behavior of the conversion function of an A-compatible database is strict, that is, the **NULL** input value is skipped. For GaussDB, you need to define the **strict** attribute of the conversion function.

- **state_data_type**

  Data type of the aggregation status value.

- **ffunc**

  Final processing function called after all the input lines have been converted, which calculates the result of aggregation. This function must accept a parameter of **state_data_type**. The output data type of the aggregation is defined as the return type of this function. If **ffunc** is not specified, the state value of the aggregation result is used as the aggregation result, and the output type is **state_data_type**.

- **initial_condition**

  Initial setting (value) of a state value. It must be a text constant value acceptable to **state_data_type**. If the parameter is not specified, the initial state value is **NULL**.

- **sort_operator**

  Sort operator used for MIN or MAX aggregation. This is just an operator name (optionally schema-qualified). This operator assumes that the input data type is the same as that of aggregation.

- **collection_func**

  Currently, this parameter does not take effect in centralized mode.

- **initial_collection_condition**

  Collects the initial setting (value) of the function status value. It must be a text constant value acceptable to **state_data_type**. If the parameter is not specified, the initial state value is **NULL**.

- **init_func**

  Initial setting function of the **sfunc** status value. It cannot return **NULL**. It must return the state_data_type type and have no input parameter. If both **init_func** and **initial_condition** are set, the return value of **init_func** is preferentially used as the initial value.

- **is_shippable**

  Specifies whether the aggregate function can be pushed down. The value can only be **true** or **false**. Currently, this parameter does not affect the pushdown of built-in aggregate functions. It affects only the behavior of user-defined aggregate functions. The default value is **false**. Currently, this parameter does not take effect in centralized mode.

Examples

```
-- Create a user-defined function.
gaussdb=# CREATE OR REPLACE FUNCTION int_add(int,int)
 returns int as $BODY$
declare
begin
 return $1 + $2;
```



```
end;
$BODY$ language plpgsql;

-- Create an aggregate function.
gaussdb=# CREATE AGGREGATE sum_add(int)
(
    sfunc = int_add,

    stype = int,

    initcond = '0'

);

-- Create a test table and add data.
gaussdb=# CREATE TABLE test_sum(a int,b int,c int);
gaussdb=# INSERT INTO test_sum VALUES(1,2),(2,3),(3,4),(4,5);

-- Execute the aggregate function.
gaussdb=# SELECT sum_add(a) FROM test_sum;
  sum_add
-----------
  10

-- Delete the aggregate function.
gaussdb=# DROP AGGREGATE sum_add(int);

-- Delete the user-defined function.
gaussdb=# DROP FUNCTION int_add(int,int);

-- Delete the test table.
gaussdb=# DROP TABLE test_sum;
```

Helpful Links

ALTER AGGREGATE and **DROP AGGREGATE**

7.12.8.11 CREATE AUDIT POLICY

Description

Creates a unified audit policy.

Precautions

- Only the user with the poladmin or sysadmin permission, or initial user has the permission to maintain audit policies.
- Before creating an audit policy, ensure that the security policy function has been enabled. That is, the masking policy takes effect only after the GUC parameter **enable_security_policy** is set to **on**.
- A system administrator or security policy administrator can access the GS_AUDITING_POLICY, GS_AUDITING_POLICY_ACCESS, GS_AUDITING_POLICY_PRIVILEGES, and GS_AUDITING_POLICY_FILTERS system catalogs to query the created audit policies.
- The audit policy name must be unique to avoid conflicts with existing policies. You can use IF NOT EXISTS to check whether the specified audit policy exists to avoid repeated creation.

NOTICE

When you use database links to perform operations on remote objects, the client initiates a database link request. The actual sender is the server, and the attributes such as the IP address of the sender are the values of the server. For details, see **DATABASE LINK**.

Syntax

```
CREATE AUDIT POLICY [ IF NOT EXISTS ] policy_name { { privilege_audit_clause | access_audit_clause }
[, ... ] [ filter_group_clause ] [ ENABLE | DISABLE ] };
```

- privilege_audit_clause
  ```
  PRIVILEGES { DDL | ALL } [ ON LABEL ( resource_label_name [, ... ] ) ]
  ```
- access_audit_clause
  ```
  ACCESS { DML | ALL } [ ON LABEL ( resource_label_name [, ... ] ) ]
  ```
- filter_group_clause
  ```
  FILTER ON { FILTER_TYPE ( filter_value [, ... ] ) } [, ... ]
  ```

Parameters

- **policy_name**

  Specifies the audit policy name, which must be unique.

  Value range: a string. It must comply with the **naming convention**.
- **resource_label_name**

  Specifies the resource label name.
- **DDL**

  Specifies the operations that are audited in the database: CREATE, ALTER, DROP, ANALYZE, COMMENT, GRANT, REVOKE, SET, and SHOW.

  If this parameter is set to **ANALYZE**, both ANALYZE and VACCUM operations are audited.
- **DML**

  Specifies the operations that are audited within the database: SELECT, COPY, DEALLOCATE, DELETE, EXECUTE, INSERT, PREPARE, REINDEX, TRUNCATE, and UPDATE.
- **ALL**

  Specifies all operations supported by the specified DDL or DML statements in the database. When the form is { DDL | ALL }, **ALL** indicates all DDL operations. When the form is { DML | ALL }, **ALL** indicates all DML operations.
- **FILTER_TYPE**

  Specifies the types of information to be filtered by the policy, including **APP**, **ROLES**, and **IP**.
- **filter_value**

  Indicates the detailed information to be filtered.
- **ENABLE|DISABLE**

  Enables or disables the unified audit policy. If **ENABLE|DISABLE** is not specified, **ENABLE** is used by default.

Examples

- Creates audit policy for executing **CREATE** on the database.

```
-- Create the adt1 policy.
gaussdb=# CREATE AUDIT POLICY adt1 PRIVILEGES CREATE;

-- View the adt1 policy.
gaussdb=# SELECT * FROM GS_AUDITING_POLICY;
 polname | polcomments |         modifydate         | polenabled
---------+-------------+----------------------------+------------
 adt1    |             | 2023-11-06 16:41:40.947417 | t

-- Check the location where the audit policy is stored.
gaussdb=# SHOW audit_directory;

-- Delete the audit policy adt1.
gaussdb=# DROP AUDIT POLICY adt1;
```

- Create an audit policy to audit only the CREATE operation performed by the **dev_audit** user.

```
-- Create user dev_audit.
gaussdb=# CREATE USER dev_audit PASSWORD '********';

-- Create the tb_for_audit table.
gaussdb=# CREATE TABLE tb_for_audit(col1 text, col2 text, col3 text);

-- Create the adt_lb0 resource label based on the tb_for_audit table.
gaussdb=# CREATE RESOURCE LABEL adt_lb0 add TABLE(public.tb_for_audit);

-- Create the adt2 audit policy for the CREATE operation on the adt_lb0 resource.
gaussdb=# CREATE AUDIT POLICY adt2 PRIVILEGES CREATE ON LABEL(adt_lb0) FILTER ON
ROLES(dev_audit);

-- Delete the audit policy adt2.
gaussdb=# DROP AUDIT POLICY adt2;

-- Delete the tb_for_audit table.
gaussdb=# DROP TABLE tb_for_audit;

-- Delete the dev_audit user.
gaussdb=# DROP USER dev_audit;
```

- Create an audit policy to audit only the SELECT, INSERT, and DELETE operations performed on the **adt_lb0** resource by user **dev_audit** using client tool gsql on the servers whose IP addresses are **10.20.30.40** and **127.0.0.0/24**, respectively.

```
-- Create a user dev_audit.
gaussdb=# CREATE USER dev_audit PASSWORD '********';

-- Create the audit policy adt3.
gaussdb=# CREATE AUDIT POLICY adt3 ACCESS SELECT ON LABEL(adt_lb0), INSERT ON
LABEL(adt_lb0), DELETE FILTER ON ROLES(dev_audit), APP(gsql), IP('10.20.30.40', '127.0.0.0/24');

-- Delete the audit policy adt3.
gaussdb=# DROP AUDIT POLICY adt3;

-- Delete the dev_audit user.
gaussdb=# DROP USER dev_audit;
```

Helpful Links

ALTER AUDIT POLICY and **DROP AUDIT POLICY**

7.12.8.12 CREATE CAST

Description

Defines a conversion.

Syntax

- Define the CAST as a conversion with function.

```
CREATE CAST (source_type AS target_type)
   WITH FUNCTION function_name (argument_type [, ...])
   [ AS ASSIGNMENT | AS IMPLICIT ];
```

- Define the CAST as a conversion without function.

```
CREATE CAST (source_type AS target_type)
   WITHOUT FUNCTION
   [ AS ASSIGNMENT | AS IMPLICIT ];
```

- Defines the CAST as a conversion with I/O.

```
CREATE CAST (source_type AS target_type)
   WITH INOUT
   [ AS ASSIGNMENT | AS IMPLICIT ];
```

Parameters

- **source_type**

  Type of the source data to be converted.

- **target_type**

  Type of the target data to be converted.

- **function_name(argument_type [, ...])**

  Function used for conversion. The function name can be modified with a schema name. If it is not modified with a schema name, the function will be found in the schema search path. The result data type of the function must match the target type of the conversion. Its parameters are discussed below.

- **WITHOUT FUNCTION**

  Indicates that the source type is a binary castable to the target type, so no function is needed to perform this conversion.

- **WITH INOUT**

  Indicates that the conversion is an I/O conversion, which is performed by calling the output function of the source data type and transferring the result to the input function of the target data type.

- **AS ASSIGNMENT**

  Indicates that the conversion can be implicitly called in assignment mode.

- **AS IMPLICIT**

  Indicates that the conversion can be implicitly called in any environment.

  A conversion implementation function can have one to three parameters. The type of the first parameter must be the same as that of the source type to be converted, or can be forcibly converted from the binary of the source type to be converted. If the second parameter exists, it must be of the integer type. It receives these type modifiers associated with the target type, or **-1** if nothing is present. If the third parameter exists, it must be of the Boolean type. If the conversion is an explicit type conversion, **true** is received. Otherwise, **false** is received.

The return type of a conversion function must be the same as the target type of the conversion, or the two types can be binary coercible.

Typically, a transformation must have different source and target data types. However, if there is a conversion implementation function with more than one parameter, it is allowed to declare a conversion with the same source and target types. This is used to represent a length enforcement function of a specific type in the system catalog. The named function is used to force a value of this type to be the type modifier value given by the second parameter.

If the source type and target type of a type conversion are different and more than one parameter is received, it indicates that only one step is required to convert one type to another and the length conversion is performed at the same time. If no such conversion is available, converting to a type that uses a type modifier involves two steps, one to convert between data types, and the other to apply a conversion specified by the modifier.

Currently, domain type conversion does not take effect. Transformations are typically targeted to the domain-related data types to which they belong.

NOTE

Cast is executed depending on the permission of the user who calls it. When invoking a cast created by others, check the execution content of the cast function to prevent the cast creator from performing unauthorized operations with the permission of the executor.

Examples

To create an assignment mapping from the double precision type to the timestamp with time zone type, use the double_to_timestamp(double precision) function.

```
-- Create a function.
gaussdb=# CREATE OR REPLACE FUNCTION double_to_timestamp(double precision) RETURNS TIMESTAMP
WITH TIME ZONE AS $$
SELECT to_timestamp($1);
$$ LANGUAGE SQL STRICT;

-- Create a type conversion.
gaussdb=# CREATE CAST(double precision AS timestamp with time zone) WITH FUNCTION
double_to_timestamp(double precision) AS IMPLICIT;

-- Delete the type conversion.
gaussdb=# DROP CAST (double precision AS timestamp with time zone);
```

Compatibility

The CREATE CAST instruction complies with the SQL standard. Except that the SQL does not have extra parameters that can be forcibly converted to binary types or implement functions.

Helpful Links

DROP CAST

7.12.8.13 CREATE CLIENT MASTER KEY

Description

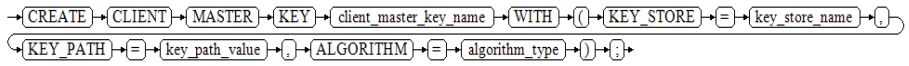
The encrypted equality query feature adopts a multi-level encryption model. The master key encrypts the column key, and the column key encrypts data. This syntax is used to create a master key object.

Precautions

- This syntax is specific to a fully-encrypted database.
- When connecting to the database, you need to enable the connection parameters of the encrypted equality query feature on the database driver side before running this syntax.
- The master key is provided by an external key manager. This syntax processes only information such as the key source and key ID. The following external key managers are supported:
  a. Huawei Cloud KMShuawei_kms
  b. Derived key from the user password or provided key user_token
- Before using this syntax, set environment variables for the external key manager on the database driver side. For details, see section "Setting Encrypted Equality Query" in *Feature Guide*.

Syntax

```
CREATE CLIENT MASTER KEY client_master_key_name WITH (KEY_STORE = key_store_name, KEY_PATH = key_path_value, ALGORITHM = algorithm_type);
```



Parameters

- **client_master_key_name**

  This parameter is used as the name of a key object. In the same namespace, the value of this parameter must be unique.

  Value range: a string. It must comply with the **naming convention**.

- **KEY_STORE**

  External key manager. For details about the value, see **Table 7-217**.

  If **KEY_STORE** is set to **user_token**, you do not need to provide the **KEY_PATH** parameter.

- **KEY_PATH**

  Each key is managed by an external key manager and the key path format varies depending on the key manager. The value is a character string. For details, see **Table 7-217**. A character string is enclosed in single or double quotation marks. If the length of a character string exceeds 64 characters, only single quotation marks can be used.

- **ALGORITHM**

  Encryption algorithm used by the key. For details about the value, see **Table 7-217**.

Table 7-217 Parameter values for different key managers

KEY_STORE	KEY_PATH	ALGORITHM
huawei_kms	Format: '{KmsApiUrl}/{*Key ID*}' Reference: 'https://kms.{*Project*}.myhuaweicloud.com/v1.0/{*Project ID*}/kms/{*Key ID*}' Example: 'https://kms.cn-north-4.myhuaweicloud.com/v1.0/00000000000000000000000000000000/kms/00000000-0000-0000-0000-000000000000'	AES_256 SM4
user_token	Users do not need to provide **KEY_PATH**.	AES_256_CBC AES_256_GCM SM4

Examples

- user_token scenario:

```
-- Decompress the installation package GaussDB-Kernel_Database version number_OS version number_64bit_Gsql.tar.gz to find the gsql_env.sh script.

-- Use the script to automatically configure the environment variables.
source gsql_env.sh

-- Connect to the database, use the privileged account, and create a user, for example, alice.
gsql -p Port number -d postgres -r
gaussdb=# CREATE USER alice PASSWORD '*******';
gaussdb=# \q

-- Connect to the database. The -C parameter must be used.
gsql -p Port number -d postgres -U alice -r -C

-- Set the user password or derived key. gsql can use password=stdin or key_token=stdin for interactive input.
-- Set the user password. The password must contain at least eight characters, including three types of the following: uppercase letters, lowercase letters, digits, and symbols.
gaussdb=> \key_info keyType=user_token,password=*********
-- Alternatively, connect to a key that meets the security strength requirements. The key is a hexadecimal code.
gaussdb=> \key_info keyType=user_token,key_token=********************************

-- Create a master key.
gaussdb=> CREATE CLIENT MASTER KEY alice_cmk WITH ( KEY_STORE = user_token , ALGORITHM = AES_256_GCM );

-- Delete the master key.
gaussdb=> DROP CLIENT MASTER KEY alice_cmk;
gaussdb=> \q

-- Connect to the database, use the privileged account, and delete user alice.
gsql -p Port number -d postgres -r
gaussdb=# DROP USER alice;
```


Helpful Links

DROP CLIENT MASTER KEY

7.12.8.14 CREATE COLUMN ENCRYPTION KEY

Description

Creates a CEK that can be used to encrypt a specified column in a table.

Precautions

- This syntax is specific to a fully-encrypted database.
- When using gsql to connect to a database server, you need to use the **-C** parameter to enable the fully-encrypted database.
- The CEK object created using this syntax can be used for column-level encryption. When defining a column in a table, you can specify a CEK object to encrypt the column.

Syntax

```
CREATE COLUMN ENCRYPTION KEY column_encryption_key_name WITH VALUES(CLIENT_MASTER_KEY = client_master_key_name, ALGORITHM = algorithm_type [, ENCRYPTED_VALUE = encrypted_value]);
```

Parameters

- **column_encryption_key_name**

  Key object name. In the same namespace, the name must be unique.

  Value range: a string. It must comply with the **naming convention**.

- **CLIENT_MASTER_KEY**

  CMK, which is used to encrypt a specified CEK. The value is the name of a CMK. The CMK object is created using the CREATE CLIENT MASTER KEY syntax.

- **ALGORITHM**

  Specifies an encryption algorithm to be used by the CEK. The value can be **AEAD_AES_256_CBC_HMAC_SHA256**, **AEAD_AES_128_CBC_HMAC_SHA256**, **AEAD_AES_256_CTR_HMAC_SHA256**, **AES_256_GCM**, or **SM4_SM3**.

  The data expansion rates of different encryption algorithms are sorted as follows: AEAD_AES_256_CTR_HMAC_SHA256 < AES_256_GCM < AEAD_AES_256_CBC_HMAC_SHA256 = AEAD_AES_128_CBC_HMAC_SHA256 = SM4_SM3. The AEAD_AES_256_CTR_HMAC_SHA256 and AES_256_GCM encryption algorithms are recommended.

  If **KEY_STORE** (master key) is set to **third_kms**, you do not need to provide the **ALGORITHM** parameter.

- **ENCRYPTED_VALUE (optional)**

  A key password specified by a user. The key password length ranges from 28 to 256 characters. The derived 28-character key meets the AES128 security requirements. If the user needs to use AES256, the key password length must be 39 characters. If the user does not specify the key password length, a 256-character key is automatically generated.

NOTICE

- SM algorithm constraints: SM2, SM3, and SM4 are Chinese national cryptography standards. To avoid legal risks, these algorithms must be used together. If you specify the SM4 algorithm to encrypt CEKs when creating a CMK, you must specify the SM3 and SM4 algorithms (SM4_SM3) to encrypt data when creating CEKs.
- Constraints on the **ENCRYPTED_VALUE** column: If the CMK generated by Huawei KMS is used to encrypt the CEK and the **ENCRYPTED_VALUE** column is used to transfer the key in the CREATE COLUMN ENCRYPTION KEY syntax, the length of the input key must be an integer multiple of 16 bytes.

Helpful Links

ALTER COLUMN ENCRYPTION KEY and **DROP COLUMN ENCRYPTION KEY**

7.12.8.15 CREATE CONVERSION

Description

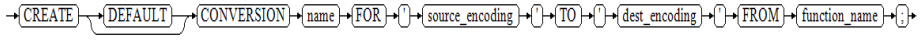
Defines a new conversion between two character set encodings. This function is for internal use only. You are advised not to use it.

Precautions

- The **DEFAULT** parameter indicates that the conversion between the source encoding and the target encoding is executed by default between the client and the server. To support this usage, bidirectional conversion must be defined, that is, both conversion from A to B and conversion from B to A are supported.
- To perform conversion, you must have the EXECUTE permission on function and the CREATE permission on the target schema.
- SQL_ASCII cannot be used for either source encoding or target encoding because the server behavior is hardwired when SQL_ASCII "encoding" is involved.
- You can remove user-defined conversions using DROP CONVERSION.

Syntax

```
CREATE [ DEFAULT ] CONVERSION name
    FOR 'source_encoding' TO 'dest_encoding' FROM function_name;
```



Parameters

- **DEFAULT**

  Specifies that the conversion is the default conversion from the source encoding to the target encoding. There should be only one default conversion for each encoding pair in a schema.

- **name**

  Specifies the name of the conversion, which can be restricted by the schema. If not restricted by a schema, the conversion is defined in the current schema. The conversion name must be unique in a schema.

- **source_encoding**

  Source encoding name.

- **dest_encoding**

  Target encoding name.

- **function_name**

  Function used for conversion. A function name can be restricted by a schema. If not, the function is found in the path. The return value of the function is the number of bytes that are successfully converted.

  The function must be in the following format:

```
conv_proc(
    integer,  -- Source encoding ID
    integer,  -- Target encoding ID
    cstring, -- Source character string (a null-terminated C character string)
    internal,-- Target (filled with a null-terminated C character string)
    integer  -- Length of the source string
) RETURNS void;
```

CAUTION

- Currently, only internal creation is supported.

7.12.8.16 CREATE DATABASE

Description

Creates a database. By default, the new database will be created only by cloning the standard system database **template0**.

Precautions

- A user who has the CREATEDB permission or a system administrator can create a database.
- It cannot be executed inside a transaction block.
- During the database creation, an error message indicating that permission denied is displayed, possibly because the permission on the data directory in the file system is insufficient. If an error message, indicating no space left on device is displayed, the possible cause is that the disk space is used up.

Syntax

```
CREATE DATABASE database_name
   [ [ WITH ] { [ OWNER [=] user_name ] |
            [ TEMPLATE [=] template ] |
            [ ENCODING [=] 'encoding' ] |
            [ LC_COLLATE [=] 'lc_collate' ] |
            [ LC_CTYPE [=] 'lc_ctype' ] |
            [ DBCOMPATIBILITY [=] 'compatibility_type' ] |
            [ TABLESPACE [=] tablespace_name ] |
```



```
    [ CONNECTION LIMIT [=] connlimit ] |
    [ DBTIMEZONE [=] 'time_zone' ] }[...] ];
```

Parameters

- **database_name**

  Specifies the database name.

  Value range: a string. It must comply with the **naming convention**.

- **OWNER [=] user_name**

  (Optional) Specifies the owner of the new database. If omitted, the default owner is the current user.

  Value range: an existing username

- **TEMPLATE [=] template**

  (Optional) Specifies a template name. That is, the template from which the database is created. GaussDB creates a database by copying data from a template database. GaussDB has two default template databases **template0** and **template1** and a default user database **postgres**.

  After the templatem feature is enabled, a template database templatem is added. Enable the templatem feature. For details, see section "Creating the M-compatible Database and User" in *M-Compatibility Developer Guide*.

  Value range: **template0** and **templatem**

- **ENCODING [=] 'encoding'**

  (Optional) Specifies the character encoding used by the database. The value can be a string (for example, **SQL_ASCII**) or an integer.

  If this parameter is not specified, the UTF-8 encoding format is used for the M-compatible database by default, and the encoding format of the template database is used for other databases by default. By default, the codes of the template databases **template0** and **template1** are related to the OS environment. The character encoding of **template1** cannot be changed. To change the encoding, use **template0** to create a database.

  Common values are **GBK**, **UTF8**, **Latin1**, and **GB18030**. The supported character sets are as follows:

Table 7-218 GaussDB character set

Name	Description	Language	Server-side Encoding	ICU Support	Number of Bytes/ Characters	Alias
BIG5	Big Five	Traditional Chinese	No	No	1–2	WIN950, Windows950
EUC_CN	Extended Unix Code-CN	Simplified Chinese	Yes	Yes	1–3	-

Name	Description	Language	Server-side Encoding	ICU Support	Number of Bytes/ Characters	Alias
EUC_JP	Extended Unix Code-JP	Japanese	Yes	Yes	1–3	-
EUC_JIS_2004	Extended Unix Code-JP, JIS X 0213	Japanese	Yes	No	1–3	-
EUC_KR	Extended Unix Code-KR	Korean	Yes	Yes	1–3	-
EUC_TW	Extended Unix Code-Taiwan, China	Traditional Chinese	Yes	Yes	1–3	-
GB18030	National standards	Chinese	Yes	No	1–4	-
GB18030_2022	National standards	Chinese	Yes	No	1–4	-
GBK	Extended national standards	Simplified Chinese	Yes	No	1–2	WIN936, Windows936
ISO_8859_5	ISO 8859-5, ECMA 113	Latin/ Cyrillic	Yes	Yes	1	-
ISO_8859_6	ISO 8859-6, ECMA 114	Latin/ Arabic	Yes	Yes	1	-
ISO_8859_7	ISO 8859-7, ECMA 118	Latin/ Greek	Yes	Yes	1	-

Name	Description	Language	Server-side Encoding	ICU Support	Number of Bytes/ Characters	Alias
ISO_8859_8	ISO 8859-8, ECMA 121	Latin/ Hebrew	Yes	Yes	1	-
JOHAB	JOHAB	Korean	No	No	1–3	-
KOI8R	KOI8-R	Cyrillic (Russian)	Yes	Yes	1	KOI8
KOI8U	KOI8-U	Cyrillic (Ukrainian)	Yes	Yes	1	-
LATIN1	ISO 8859-1, ECMA 94	Western European languages	Yes	Yes	1	ISO88591
LATIN2	ISO 8859-2, ECMA 94	Central European languages	Yes	Yes	1	ISO88592
LATIN3	ISO 8859-3, ECMA 94	South European languages	Yes	Yes	1	ISO88593
LATIN4	ISO 8859-4, ECMA 94	North European languages	Yes	Yes	1	ISO88594
LATIN5	ISO 8859-9, ECMA 128	Turkish	Yes	Yes	1	ISO88599
LATIN6	ISO 8859-10, ECMA 144	Germanic languages	Yes	Yes	1	ISO885910

Name	Description	Language	Server-side Encoding	ICU Support	Number of Bytes/ Characters	Alias
LATIN7	ISO 8859-13	Baltic languages	Yes	Yes	1	ISO885913
LATIN8	ISO 8859-14	Celtic languages	Yes	Yes	1	ISO885914
LATIN9	ISO 8859-15	LATIN1 with Euro and accents	Yes	Yes	1	ISO885915
LATIN10	ISO 8859-16, ASRO SR 14111	Romanian	Yes	No	1	ISO885916
MULE_INTERNAL	Mule internal code	Multilingual Emacs	Yes	No	1–4	-
SJIS	Shift JIS	Japanese	No	No	1–2	Mskanji, ShiftJIS, WIN932, Windows932
SHIFT_JIS_2004	Shift JIS, JIS X 0213	Japanese	No	No	1–2	-
SQL_ASCII	Unspecified (see the text)	Any	Yes	No	1	-
UHC	Unified Hangul Code	Korean	No	No	1–2	WIN949, Windows949
UTF8	Unicode, 8-bit	All	Yes	Yes	1–4	Unicode
WIN866	Windows CP866	Cyrillic	Yes	Yes	1	ALT
WIN874	Windows CP874	Thai	Yes	No	1	-

Name	Description	Language	Server-side Encoding	ICU Support	Number of Bytes/ Characters	Alias
WIN1250	Windows CP1250	Central European languages	Yes	Yes	1	-
WIN1251	Windows CP1251	Cyrillic	Yes	Yes	1	WIN
WIN1252	Windows CP1252	Western European languages	Yes	Yes	1	-
WIN1253	Windows CP1253	Greek	Yes	Yes	1	-
WIN1254	Windows CP1254	Turkish	Yes	Yes	1	-
WIN1255	Windows CP1255	Hebrew	Yes	Yes	1	-
WIN1256	Windows CP1256	Arabic	Yes	Yes	1	-
WIN1257	Windows CP1257	Baltic languages	Yes	Yes	1	-
WIN1258	Windows CP1258	Vietnamese	Yes	Yes	1	ABC, TCVN, TCVN5712, VSCII
ZHS16GBK	Extended national standards	Simplified Chinese	Yes	No	1–2	-

CAUTION

- Note that not all client APIs support the preceding character sets.
- The SQL_ASCII setting performs quite differently from other settings. If the character set of the server is SQL_ASCII, the server interprets the byte values 0 to 127 according to the ASCII standard. The byte values 128 to 255 are regarded as the characters that cannot be parsed. If this parameter is set to SQL_ASCII, no code conversion occurs. Therefore, this setting is not basically used to declare the specified encoding used, because this declaration ignores the encoding. In most cases, if you use any non-ASCII data, it is unwise to use the SQL_ASCII setting because the database will not be able to help you convert or verify non-ASCII characters.

NOTICE

- The character set encoding of the new database must be compatible with the local settings (**LC_COLLATE** and **LC_CTYPE**).
- When the specified character encoding set is **GBK** or **ZHS16GBK**, some uncommon Chinese characters cannot be directly used as object names. This is because the byte encoding overlaps with the ASCII characters @A-Z[\]^_`a-z{|} when the second byte of the GBK ranges from 0x40 to 0x7E. **@[\]^_'{|}** is an operator in the database. If it is directly used as an object name, a syntax error will be reported. For example, the GBK hexadecimal code is **0x8240**, and the second byte is **0x40**, which is the same as the ASCII character @. Therefore, the character cannot be used as an object name. If you need to use this function, you can add double quotation marks ("") to avoid this problem when creating and accessing objects.
- If the client code is A and the server code is B, the conversion between encoding formats A and B must exist in the database. For details about encoding format conversion supported by the database, see the system catalog **PG_CONVERSION**. (If the encoding format cannot be converted, it is recommended that the encoding format on the client be the same as that on the server. You can change the encoding format on the client by setting the **client_encoding** parameter.)
- If you want to specify both the database character set encoding and client encoding to GB18030_2022, ensure that the client OS supports GB18030_2022. If the GB18030 character set versions are incompatible with each other, data inconsistency may occur. In addition, if historical data needs to be switched to the GB18030_2022 database, follow the database switching process to migrate data.

- **LC_COLLATE [=] 'lc_collate'**

  (Optional) Specifies the character set used by the new database. For example, set this parameter by using **lc_collate = 'zh_CN.gbk'**.

  The use of this parameter affects the sort order of strings (for example, the order of using **ORDER BY** for execution and the order of using indexes on text columns). By default, the character set of the template database is used. This parameter does not take effect for the M-compatible database.

  Value range: character sets supported by the OS.

- **LC_CTYPE [=] 'lc_ctype'**

  (Optional) Specifies the character class used by the new database. For example, set this parameter by using **lc_ctype = 'zh_CN.gbk'**. The use of this parameter affects the classification of characters, such as uppercase letters, lowercase letters, and digits. By default, the character classification of the template database is used. This parameter does not take effect for the M-compatible database.

  Value range: character classes supported by the OS.

  NOTE

  - The value ranges of **lc_collate** and **lc_ctype** depend on the character sets supported by the local environment. For example, in the Linux OS, you can run the **locale -a** command to obtain the list of character sets supported by the OS. When using the **lc_collate** and **lc_ctype** parameters, you can select the required character sets and character classes.
  - If you want to set the character encoding set to GB18030_2022, ensure that the value ranges of **lc_collate** and **lc_ctype** are the same as those of GB18030.
  - If you want to set the character encoding set to ZHS16GBK, ensure that the value ranges of **lc_collate** and **lc_ctype** are the same as those of GBK character set.

- **DBCOMPATIBILITY [=] compatibility_type**

  (Optional) Specifies the compatible database type. The default database is an O-compatible database.

  Value range: **A**, **B**, **C**, **PG**, and **M**, indicating O-, MY-, TD-, POSTGRES-, and M-compatible databases, respectively.

  NOTE

  - For A compatibility, the database treats empty strings as **NULL** and replaces **DATE** with **TIMESTAMP(0) WITHOUT TIME ZONE**.
  - When a character string is converted to an integer, if the input is invalid, the input will be converted to 0 due to B compatibility, and an error will be reported due to other compatibility issues.
  - For PG compatibility, CHAR and VARCHAR are counted by character. For other compatibility types, they are counted by byte. For example, for the UTF-8 character set, CHAR(3) can store three Chinese characters in PG compatibility scenarios, but can store only one Chinese character in other compatibility scenarios.
  - For details about database features in M-compatible mode, see *M-Compatibility Development Guide*.

- **TABLESPACE [=] tablespace_name**

  (Optional) Specifies the tablespace of the database.

  Value range: an existing tablespace name

- **CONNECTION LIMIT [=] connlimit**

  (Optional) Specifies the maximum number of concurrent connections that can be made to the new database.

  NOTICE

  - The system administrator is not restricted by this parameter.
  - **connlimit** is calculated separately for each primary database node. Number of connections of the database = Value of **connlimit** x Number of normal primary database nodes.

Value range: an integer in the range [–1, 2^{31} – 1]. The default value is **-1**, indicating that there is no limit.

The restrictions on character encoding are as follows:

- If the locale is set to **C** (or **POSIX**), all encoding types are allowed. For other locale settings, the character encoding must be the same as that of the locale.
- If the character encoding mode is SQL_ASCII and the modifier is an administrator, the character encoding mode can be different from the locale setting.
- The encoding and region settings must match the template database, except that **template0** is used as a template. This is because other databases may contain data that does not match the specified encoding, or may contain indexes whose sorting order is affected by **LC_COLLATE** and **LC_CTYPE**. Copying this data will invalidate the indexes in the new database. **template0** does not contain any data or indexes that may be affected.
- **DBTIMEZONE [=] time_zone**

  Specifies the time zone of the new database. For example, you can set this parameter by setting **DBTIMEZONE** to **'+00:00'**. This parameter affects the time zone of the new database. The PRC is used by default.

  Prerequisites: The current database is compatible with database A, **a_format_version** is set to **'10c'**, and **a_format_dev_version** is set to **'s2'**.

  Value range: name and abbreviation of the time zone supported by the OS, or the timestamp ranges from –15:59 to +15:00.

Examples

```
-- Create user jim.
gaussdb=# CREATE USER jim PASSWORD '********';

-- Create the GBK-encoded database testdb1.
gaussdb=# CREATE DATABASE testdb1 ENCODING 'GBK' template = template0;
-- View information about the testdb1 database.
gaussdb=#SELECT datname,pg_encoding_to_char(encoding) FROM pg_database WHERE datname =
'testdb1';
 datname | pg_encoding_to_char
---------+---------------------
 testdb1 | GBK
(1 row)
-- Create the A-compatible database testdb2 and specify jim as the owner.
gaussdb=# CREATE DATABASE testdb2 OWNER jim DBCOMPATIBILITY = 'A';
-- View the information about testdb2.
gaussdb=# SELECT t1.datname,t2.usename,t1.datcompatibility
                FROM pg_database t1,pg_user t2
                WHERE t1.datname = 'testdb2' AND t1.datdba=t2.usesysid;
 datname | usename | datcompatibility
---------+---------+------------------
 testdb2 | jim     | A
(1 row)
-- Switch to the A-compatible database testdb2 and set session parameters.
gaussdb=# \c testdb2
testdb2=# SET a_format_version='10c';
testdb2=# SET a_format_dev_version='s2';
-- Create the A-compatible database and specify the time zone.
testdb2=# CREATE DATABASE testdb3 DBCOMPATIBILITY 'A' DBTIMEZONE='+08:00';
-- View the information about testdb3.
testdb2=# SELECT datname,datcompatibility,dattimezone FROM pg_database WHERE datname = 'testdb3';
 datname | datcompatibility | dattimezone
---------+------------------+-------------
 testdb3 | A                | +08:00
(1 row)
```



```
-- Switch to the initial database.
testdb2=# \c postgres
-- Delete the database.
gaussdb=# DROP DATABASE testdb1;
gaussdb=# DROP DATABASE testdb2;
gaussdb=# DROP DATABASE testdb3;
-- Delete the user.
gaussdb=# DROP USER jim;
```

Helpful Links

ALTER DATABASE and **DROP DATABASE**

Suggestions

- **create database**

  Database cannot be created in a transaction.

- **ENCODING LC_COLLATE LC_CTYPE**

  If the new database encoding does not match the template database (SQL_ASCII) (**'GBK'**, **'UTF8'**, **'LATIN1'**, **'GB18030_2022'**, **'ZHS16GBK'**, or **'GB18030'**), **template [=] template0** must be specified.

7.12.8.17 CREATE DATABASE LINK

Description

Creates a database link object. For details about database links, see **DATABASE LINK**.

Precautions

- The database link feature can be used only in A-compatible versions.
- Do not use database links to connect to the initial user.
- Initial users are not allowed to create, modify, or delete database links.
- If the upgrade is not committed, the database link cannot be created.
- If the CURRENT_USER is used or the CONNECT TO connection string is omitted, the initial username and empty password of the current database are used for connection, resulting in connection failure.

Syntax

```
CREATE [ PUBLIC ] DATABASE LINK dblink
  [ CONNECT TO { CURRENT_USER | 'user' IDENTIFIED BY 'password' } ] [ USING ( option 'value' [...]) ];
```

Parameters

- **PUBLIC**

  Creates a public database link visible to all users. If this clause is omitted, the database link is private and available only to the current user.

- **dblink**

  Specifies the name of the database link to be created.

- **user**

Specifies the username of the user for connecting to the remote end using the created database link.

- **password**

  Specifies the user password for connecting to the remote end using the created database link.

- **CURRENT_USER**

  Uses the initial username and empty password of the current database for connection.

- **USING (option 'value' [, ...])**

  Specifies parameters such as the IP address, port number, and remote database name of the database to be connected. The supported options are as follows:

  - **host**: specifies the IP addresses to be connected. IPv6 addresses are not supported. Multiple IP addresses can be specified using character strings separated by commas (,). Currently, encrypted databases, SSL settings, and certificate authentication are not supported. If no IP address is specified, this parameter is left empty by default.
  - **port**: specifies the port number for connection. If this parameter is not specified, the default value **5432** is used.
  - **dbname**: specifies the name of the database to be connected. If this parameter is not specified, the username used for connecting to the remote end is used by default.
  - **fetch_size**: specifies the amount of data obtained from the remote end each time. The value of **fetch_size** ranges from 0 to 2147483647. The default value is **100**.

NOTICE

- You can write only part of the preceding options in the brackets after USING.
- If the keyword USING is not written, the content in the brackets is not written as well.
- When a database link is created, the system does not check whether the connection is successful. If related keywords are missing, an error may be reported.
- Do not use 127.0.0.1 or local host for the **host** parameter. Otherwise, the connection fails.

Examples

```
-- Create a user with the system administrator permission.
gaussdb=#  CREATE USER user1 WITH SYSADMIN PASSWORD '********';
gaussdb=#  SET ROLE user1 PASSWORD '********';

-- Create a private database link.
gaussdb=#  CREATE DATABASE LINK private_dblink CONNECT TO 'user1' IDENTIFIED BY '********' USING
(host '192.168.11.11',port '54399',dbname 'db01');

-- Delete the private database link.
gaussdb=#  DROP DATABASE LINK private_dblink;
```



```
-- Create a public database link.
gaussdb=#  CREATE PUBLIC DATABASE LINK public_dblink CONNECT TO 'user1' IDENTIFIED BY '********'
USING (host '192.168.11.11',port '54399',dbname 'db01');

-- Delete the public database link.
gaussdb=#  DROP PUBLIC DATABASE LINK public_dblink;

-- Delete the created user.
gaussdb=#  RESET ROLE;
gaussdb=#  DROP USER user1;
```

Helpful Links

ALTER DATABASE LINK and **DROP DATABASE LINK**

7.12.8.18 CREATE DIRECTORY

Description

CREATE DIRECTORY creates a directory. The directory defines an alias for a path in the server file system and is used to store data files used by users. Users can read and write these files through the dbe_file advanced package.

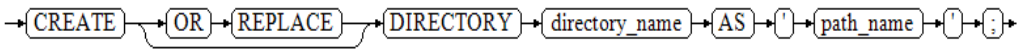
The read and write permissions for the directory can be granted to specified users to provide permission control for dbe_file.

Precautions

- When **enable_access_server_directory** is set to **off**, only the initial user is allowed to create the directory object. When **enable_access_server_directory** is set to **on**, users with the SYSADMIN permission and users inherited the built-in role permission gs_role_directory_create can create directory objects.
- By default, the user who creates a directory has the read and write permissions on the directory.
- The default owner of a directory is the user who creates the directory.
- A directory cannot be created for the following paths:
  - The path contains special characters.
  - The path is a relative path.
- The following validity check is performed during directory creation:
  - Check whether the path exists in the OS. If it does not exist, a message is displayed, indicating the potential risks.
  - Check whether the database initial user **omm** has the R/W/X permissions for the OS path. If the user does not have all the permissions, a message is displayed, indicating the potential risks.
- Ensure that the path is the same on all the nodes in the database. Otherwise, the path may fail to be found on some nodes when the directory is used.
- You can view existing directory objects in the pg_directory table.

Syntax

```
CREATE [OR REPLACE] DIRECTORY directory_name
AS 'path_name';
```

Parameters

- **directory_name**

  Specifies the name of a directory.

  Value range: a string. It must comply with the **naming convention**.

- **path_name**

  Specifies the OS path for which a directory is to be created.

  Value range: a valid OS path

Examples

```
-- Create a directory object.
gaussdb=# CREATE OR REPLACE DIRECTORY  dir  AS '/tmp/';

-- View the created directory object.
gaussdb=# select * from pg_directory;
 dirname | owner | dirpath | diracl
---------+-------+---------+--------
 dir     |    10 | /tmp    |
(1 row)

-- Delete the directory object.
gaussdb=# DROP DIRECTORY dir;
```

Helpful Links

ALTER DIRECTORY and **DROP DIRECTORY**

7.12.8.19 CREATE EVENT

Description

Creates a scheduled event. The scheduled event is a special object defined in the database. It is associated with tables or other database objects. When specified event conditions are met, the scheduled event automatically triggers the execution of pre-defined operations or statements.

Precautions

- Operations related to scheduled events are supported only when **sql_compatibility** is set to **'B'**.
- A user without the sysadmin permission must obtain the permission from the user who has the sysadmin permission to create, modify or delete the scheduled event. The operation permissions of the scheduled event are the same as those of creating scheduled events for the advanced package DBE_SCHEDULER.
- Currently, the interval expression of a scheduled event is compatible with the syntax of floating-point number, for example, interval 0.5 minute. However, the floating-point number is rounded up during calculation. Therefore, you are advised not to use the floating-point number for the interval.

- Scheduled events with the same name are not supported in the same database.
- The statements to be executed in a scheduled event are any SQL statements except security-related operations. However, some statements with restrictions fail to be executed. For example, a database cannot be created by using composite statements.
- The security-related operations are as follows.
  - Use the encryption function.
  - Create and set users and groups.
  - Connect to a database.
  - Encrypt the function.
- The definer fails to be specified for a scheduled event in the following scenarios:
  - The user who operates the scheduled event does not have the sysadmin permission.
  - The current user is inconsistent with the specified definer.
    - An initial user is specified as the definer.
    - An O&M administrator or monitoring administrator is specified as the definer.
    - The parameter **enableSeparationOfDuty** is set to **on** to enable the separation of duties.

Syntax

```
CREATE
    [DEFINER = user]
EVENT
    [IF NOT EXISTS]
    event_name
    ON SCHEDULE schedule
    [ON COMPLETION [NOT] PRESERVE]
    [ENABLE | DISABLE | DISABLE ON SLAVE]
    [COMMENT 'string']
    DO event_body;
```

- schedule

  ```
  {
      AT timestamp [+ INTERVAL interval] ...
    | EVERY interval
      [STARTS timestamp [+ INTERVAL interval] ...]
      [ENDS timestamp [+ INTERVAL interval] ...]
  }
  ```

- interval

  ```
  quantity {YEAR | MONTH | DAY | HOUR | MINUTE | SECOND |
            YEAR_MONTH | DAY_HOUR | DAY_MINUTE |
            DAY_SECOND | HOUR_MINUTE | HOUR_SECOND |
            MINUTE_SECOND}
  ```

Parameters

- **DEFINER**

  Specifies the permission for the scheduled event statement to be executed during execution. By default, the permission of the user who creates the

scheduled event is used. When a definer is specified, the permission of the specified user is used.

Only users with the sysadmin permission can specify the definer.

- **ON SCHEDULE**

  Specifies the time when a scheduled event is executed. A scheduled event can be set to be executed once or multiple times through SCHEDULE.

  - **AT timestamp [+ INTERVAL interval]** indicates that the scheduled event is executed only once at **timestamp [+ INTERVAL interval]**.
  - **EVERY interval** indicates that the scheduled event is executed at the interval specified by **interval**.

    - **STARTS timestamp [+ INTERVAL interval]** indicates that user can specify the start time for a scheduled event that can be executed repeatedly. That is, the scheduled event is executed since **timestamp [+ INTERVAL interval]**. If this parameter is left blank, the task is executed from the current time by default.
    - **ENDS timestamp [+ INTERVAL interval]** indicates that user can specify the end time for a scheduled event that can be executed repeatedly. That is, the scheduled event ends at **timestamp [+ INTERVAL interval]**. If this parameter is left empty, the default value **3999-12-31 16:00:00** is used.

- **INTERVAL**

  Specifies the interval. The value of **INTERVAL** consists of a number and a time unit specified by **quantity**, for example, **1 YEAR**.

- **ON COMPLETION [NOT] PRESERVE**

  Once a transaction is complete, the scheduled event is deleted from the system catalog immediately by default. You can overwrite the default behavior by setting **ON COMPLETION PRESERVE**.

- **ENABLE | DISABLE | DISABLE ON SLAVE**

  The scheduled event is in the **ENABLE** state by default after it is created. That is, the statement to be executed is executed immediately at the specified time. You can use the keyword **DISABLE** to change the **ENABLE** state. The performance of **DISABLE ON SLAVE** is the same as that of **DISABLE**.

- **COMMENT**

  You can add comments to the scheduled event. The comments can be viewed in the **GS_JOB_ATTRIBUTE** table.

- **DO**

  Specifies the statement to be executed for a scheduled event.

Examples

```
-- Create and switch to the test database.
gaussdb=# CREATE DATABASE test_event WITH DBCOMPATIBILITY = 'b';
gaussdb=# \c test_event

-- Create a table.
test_event=# CREATE TABLE t_ev(num int);

-- Create a scheduled event that is executed once every five seconds. After the event is executed, the scheduled event is automatically deleted.
test_event=# CREATE EVENT IF NOT EXISTS event_e1 ON SCHEDULE AT sysdate + interval 5 second DO
```



```
INSERT INTO t_ev VALUES(0);

-- Query the table five seconds later.
test_event=# SELECT * FROM t_ev;
 num
-----
   0
(1 row)

-- Create a scheduled event that is executed every minute.
test_event=# CREATE EVENT IF NOT EXISTS event_e2 ON SCHEDULE EVERY 1 minute DO INSERT INTO
t_ev VALUES(1);

-- Query the table every one minute. A new record is displayed.
test_event=# SELECT * FROM t_ev;
 num
-----
   0
   1
   1
(3 rows)

-- Delete the scheduled event.
test_event=# DROP EVENT event_e2;

-- Drop the table.
test_event=# DROP TABLE t_ev;

-- Switch back to the initial database and delete the test database. Replace postgres with the actual
database name.
test_event=#\c postgres
gaussdb=# DROP DATABASE test_event;
```

NOTICE

- If a scheduled event fails to be executed after being created, you can view the failure cause in the SHOW EVENTS or PG_JOB table.
- When operations related to user passwords (such as creating weak passwords) are performed in the statements to be executed for a scheduled event, system catalog records the password in plaintext. Therefore, you are advised not to perform operations related to user passwords in the statements to be executed for the scheduled event.

Helpful Links

ALTER EVENT, **DROP EVENT**, and **SHOW EVENTS**

7.12.8.20 CREATE EXTENSION

NOTICE

The extended function is for internal use only. You are advised not to use it.

Description

Installs an extension.

Precautions

- Before running the **CREATE EXTENSION** command to load an extension into the database, you must install the support file for the extension first.
- The **CREATE EXTENSION** command installs a new extension to a database. Ensure that no extension with the same name has been installed.
- Installing an extension means to execute an extended script file that creates an SQL entity, such as a function, data type, operator, and index-supported method.
- Installing an extension requires the same permissions as creating its component objects. For most extensions, this means that the superuser or database owner's permissions are required. For subsequent permission checks and entities created by the extension script, the role that runs the **CREATE EXTENSION** command becomes the owner of the extension.
- During the execution of CREATE EXTENSION, if the database contains database objects with the same name as that in the EXTENSION, such as packages, synonyms, operators, directories, functions, stored procedures, views, and tables, the execution will fail.
- Do not directly create extensions for the database to avoid unexpected errors and incompatibility problems after the upgrade. To create an extension, set **enable_extension** to **true**.
- During the execution of **CREATE EXTENSION**, if the GUC parameter **enable_object_special_character** is set to **off** and "@extschema@" is used in the extension script file, the value of the **schema** parameter in the extension support file cannot contain any special characters in ["$'\].

Syntax

```
CREATE EXTENSION [ IF NOT EXISTS ] extension_name
[ WITH ] [ SCHEMA schema_name ]
[ VERSION version ]
[ FROM old_version ];
```

Parameters

- **IF NOT EXISTS**

  If an extension with the same name exists in the system, no error is reported. However, a message is displayed. Note that this parameter does not ensure that the existing extensions of the system are the same as those created by the script.

- **extension_name**

  Specifies the name of the extension to be installed. The database creates the extension by using the information in the **SHAREDIR/extension/extension_name.control** file.

- **schema_name**

  The extension instance is installed in this schema, and the extended content can be reinstalled. The specified schema must exist. If it is not specified, the extended control file does not specify a schema either. In this case, the default schema is used.

> **CAUTION**
>
> Extensions do not belong to any schema (no restriction is posed on the name of extensions within the scope of a database), but an extension instance belongs to a schema.

- **version**

  Version of an extension to be installed, which can be written as an identifier or a string. The default version is specified in the extended control file.

- **old_version**

  If you want to upgrade the content that is not contained in the **old style** module, you need to specify **FROM old_version**. This option makes **CREATE EXTENSION** run an installation script to install new content into the extension instead of creating an entity. Note that **SCHEMA** specifies the schema that includes these existing entities.

Examples

Install an extension in the current database. For example, install **security_plugin**:

```
-- Before installing the extension, set enable_extension to true.
gaussdb=# SET enable_extension = true;

-- Install the extension.
gaussdb=# CREATE EXTENSION IF NOT EXISTS security_plugin;

-- Delete the extension.
gaussdb=# DROP EXTENSION security_plugin;
```

Helpful Links

ALTER EXTENSION and **DROP EXTENSION**

7.12.8.21 CREATE FOREIGN DATA WRAPPER

Description

Creates a foreign data wrapper. The user who created the foreign data wrapper becomes its owner.

Precautions

- The name of the foreign data wrapper must be unique in the database.
- Only initial users and system administrators can create foreign data wrappers.

Syntax

```
CREATE FOREIGN DATA WRAPPER name
[ HANDLER handler_function | NO HANDLER ]
[ VALIDATOR validator_function | NO VALIDATOR ]
[ OPTIONS ( option 'value' [, ... ] ) ];
```


Parameters

- **name**

  Specifies the name of the foreign data wrapper to be created.

- **HADNLER handler_function**

  handler_function is the name of a registered function used to retrieve execution functions for foreign tables. The handler function must have no arguments, and its return type must be fdw_handler.

  It is possible to create foreign data wrappers without processor functions, but foreign tables that use such wrappers can only be declared and cannot be accessed.

- **VALIDATOR validator_function**

  validator_function is the name of a registered function used to check the common options provided to the foreign data wrapper, as well as the options for the foreign server, user mapping, and foreign table that use the foreign data wrapper. If there is no validator function or **NO VALIDATOR** is declared, options are not checked at creation time (foreign data wrappers may ignore or reject invalid option descriptions at runtime, depending on the implementation). The validator function must accept two arguments: one is of text[] type, which will contain an array of options stored in the system directory, and the other is of OID type, which will be the OID of the system directory that contains the options. The return type is ignored. The function should report invalid options using the ereport(ERROR) function.

- **OPTIONS (option 'value' [, ...])**

  This clause declares options for the new foreign data wrapper. The allowed option names and values are specific to each foreign data wrapper and are validated by the validator function of the foreign data wrapper. The option name must be unique.

Examples

```
-- Create a useless foreign data wrapper dummy.
gaussdb=# CREATE FOREIGN DATA WRAPPER dummy;

-- Create a foreign data wrapper file with the handler function file_fdw_handler.
gaussdb=# CREATE FOREIGN DATA WRAPPER file HANDLER file_fdw_handler;

-- Create a foreign data wrapper mywrapper with some options.
gaussdb=# CREATE FOREIGN DATA WRAPPER mywrapper OPTIONS (debug 'true');
```

Helpful Links

ALTER FOREIGN DATA WRAPPER and **DROP FOREIGN DATA WRAPPER**

7.12.8.22 CREATE FUNCTION

Description

Creates a function.

Precautions

- If the parameters or return values of a function have precision, the precision is not checked.

- When creating a function, you are advised to explicitly specify the schemas of tables in the function definition. Otherwise, the function may fail to be executed.
- When a stored procedure is created, a write lock is added only to the CREATE stored procedure or package, and a read lock is added only to the functions and packages on which the functions depend during compilation and execution.
- **current_schema** and **search_path** specified by SET during function creation are invalid. **search_path** and **current_schema** before and after function execution should be the same.
- If a function has output parameters, the GUC parameter **set behavior_compat_options** must be set to **'proc_outparam_override'** for the output parameters to take effect. When the function is called using SELECT or CALL, an actual parameter must be provided in the position of the output parameter. Otherwise, the function fails to be called.
- Only the functions compatible with PostgreSQL or those with the **PACKAGE** attribute can be overloaded. After **REPLACE** is specified, a new function is created instead of replacing a function if the number of parameters, parameter type, or return value is different.
- You cannot create overloaded functions with different formal parameter names (the function name and parameter list type are the same).
- You cannot create a function that has the same name and parameter list as a stored procedure.
- Formal parameters cannot be overloaded if only the custom ref cursor type is different from the sys_refcursor type.
- Function overloading is not supported if only the returned data types are different.
- Function overloading is not supported if only the default values are different.
- When an overloaded function is called, the variable type must be specified.
- A-compatible functions are created for A-compatible databases and PG-compatible functions are created for PG-compatible databases. Hybrid creation is not recommended.
- If a function supports overloading, you need to add the keyword PACKAGE.
- If an undeclared variable is used in a function, an error is reported when the function is called.
- You can use the SELECT statement to specify different parameters using identical functions. The syntax does not support calling identical functions without the **PACKAGE** attribute.
- When you create a function, you cannot insert other agg functions out of the avg function or other functions.
- By default, the permissions to execute new functions are granted to PUBLIC users. For details, see **GRANT**. By default, a user inherits the permissions of the PUBLIC role. Therefore, the user has the permission to execute a function and view the definition of the function. In addition, to execute the function, the user must have the USAGE permission on the schema to which the function locates. You can revoke the default execution permissions from the PUBLIC role when creating a function and grant the function execution permission to users as needed. To avoid the time window during which new

functions can be accessed by all users, create functions and set function execution permissions in a transaction. After the database object isolation attribute is enabled, common users can view only the definitions of functions that they have permission to execute.

- When functions without parameters are called inside another function, you can omit brackets and call functions using their names directly.
- When other functions with output parameters are called in a function and an assignment expression, set the GUC parameter **set behavior_compat_options** to **'proc_outparam_override'**, define variables of the same type as the output parameters in advance, and use the variables as output parameters to call other functions with output parameters for the output parameters to take effect. Otherwise, the output parameters of the called functions will be ignored.
- Oracle-compatible functions support viewing, exporting, and importing parameter comments.
- Oracle-compatible functions support viewing, exporting, and importing comments between IS/AS and plsql_body.
- Users granted with the CREATE ANY FUNCTION permission can create or replace functions in the user schemas.
- The default permission on a function is SECURITY INVOKER. To change the permission to SECURITY DEFINER, set the GUC parameter **behavior_compat_options** to **'plsql_security_definer'**.
- If the parameter **set behavior_compat_options** is not set to **proc_outparam_override**, the **OUT** and **IN OUT** output parameters of the function directly called by an anonymous block or stored procedure cannot be of the composite type, and the return value is used as the first value of the **OUT** output parameter. As a result, the invoking fails. To correctly use the **OUT** and **IN OUT** output parameters, set the parameter **set behavior_compat_options** to **proc_outparam_override**. For details, see **Examples**.
- For PL/SQL functions, after **behavior_compat_options** is set to **'proc_outparam_override'**, the behaviors of **out** and **inout** change. In the functions, **return**, **out**, and **inout** can be returned at the same time. Before the parameter is set to **'proc_outparam_override'**, only **return** is returned. For details, see **Examples**.
- For PL/SQL functions, after **behavior_compat_options** is set to **'proc_outparam_override'**, the restrictions are as follows:
  a. If a function with the **out/inout** parameter already exists in the same schema or package, you cannot create another function with the same name with the **out/inout** parameter.
  b. The **out** parameter must be added no matter whether the SELECT or CALL statement is used to call a stored procedure.
  c. In some scenarios, functions cannot be used in expressions (compared with those before the parameter is enabled), for example, left assignment in a stored procedure and the call function. For details, see **Examples**.
  d. Functions without return cannot be called. The perform function can be used to call functions.
  e. When a function is called in a stored procedure, **out/inout** cannot be set to a constant. For details, see **Examples**.

f. After the GUC parameter **behavior_compat_options** is set to **'proc_outparam_override'**, if the return type of the function is setof, the output parameter will not take effect.

- When a function is created, it depends on an undefined object. If **behavior_compat_options** is set to **'plpgsql_dependency'**, the creation can be executed and a warning message is displayed. If **behavior_compat_options** is not set to **'plpgsql_dependency'**, the creation cannot be executed.
- When **behavior_compat_options** is set to **'plpgsql_dependency'**, if function A is called in the function and function B is contained in the input and output parameters of function A, function B will not establish a dependency. For example, **functionA(functionB())**. **gs_dependencies** only creates dependency with function A.
- If a view directly depends on an O-style function and the **behavior_compat_options** parameter is set to **'plpgsql_dependency'**, the view can be accessed when the function is created again. However, if the **behavior_compat_options** parameter is not set to **'plpgsql_dependency'**, the view cannot be accessed.
- When creating a function, you cannot use the function itself as the default value of input parameter.
- The function with **OUT** parameter cannot be called by SQL statement.
- The function with **OUT** parameter cannot be called by SELECT INTO syntax.
- Functions with **OUT** parameters cannot be called in nested mode.

  Example:

  ```
  b := func(a,func(c,1));
  ```

  Should be changed to:

  ```
  tmp := func(c,1); b := func(a,tmp);
  ```

- When a function is created, the type of the return value of the function is not checked.
- If a function with the definer specified is created in a schema of another user, the function will be executed by another user, which may cause unauthorized operations. Therefore, exercise caution when performing this operation.
- In the schema of the O&M administrator, only the initial user, system administrator, and schema owner can create objects. The schema that does not allow other users to create and modify objects is the schema of the O&M administrator.
- If the **out** parameter is used as the output parameter in an expression, the expression does not take effect in the following scenarios: (a) The execute immediate sqlv using func syntax is used to execute a function. (b) The select func into syntax is used to execute a function. (c) DML statements such as INSERT and UPDATE are used to execute a function. (d) The select where a=func() statement is used. (e) When a function with the **out** output parameter is used as an input parameter, that is, **fun (func (out b), a)**, the **out b** parameter does not take effect.
- In the RETURN statement of a function, when the constructor of the composite type is called, if the actual return type is different from the defined return type, the result can be implicitly converted to the defined return type. Cross-schema call is supported, for example: **RETURN schema.record;**. Cross-

database call is not supported, for example, **RETURN package.schema.record;**. In the RETURN statement of a function, when the function is called, the function with the OUT parameter in an expression of an arithmetic operation is not supported, for example, **RETURN func(c out) + 1**.

- When the complex function is called, for example, func(x).a, a composite type is returned. Cross-schema call is supported, but the database.schema.package.func(x).b call is not supported.
- When a stored procedure with the out parameter is called, you can set the GUC parameter **set behavior_compat_options** to **'proc_outparam_transfer_length'** to transfer the parameter length. The specifications and constraints are as follows:
  a. The following types are supported: CHAR(n), CHARACTER(n), NCHAR(n), VARCHAR(n), VARYING(n), VARCHAR2(n), and NVARCHAR2(n).
  b. If the out parameter does not take effect (for example, **perform**), the length does not need to be transferred.
  c. The following types do not support precision transfer: NUMERIC, DECIMAL, NUMBER, FLOAT, DEC, INTEGER, TIME, TIMESTAMP, INTERVAL, TIME WITH TIME ZONE, TIMESTAMP WITH TIME ZONE, TIME WITHOUT TIME ZONE, and TIMESTAMP WITHOUT TIME ZONE.
  d. The parameter length can be transferred regardless of whether the GUC parameter **set behavior_compat_options** is set to **proc_outparam_override**.
  e. If you need to transfer the length of elements of the collection type and the length of elements of the array type nested by the collection type, you need to set the GUC parameter **behavior_compat_options** to **tableof_elem_constraints**.
- Functions contain syntax and functions that use GUC parameters to control features. If GUC parameters are modified in a session, the result may be inconsistent with the expected result. If the parameters are modified, the function may retain the behavior before the modification. Therefore, exercise caution when modifying GUC parameters.

Syntax

- Syntax (compatible with PostgreSQL) for creating a user-defined function:

```
CREATE [ OR REPLACE  ] FUNCTION function_name
   [ ( [  { argname [ argmode  ] argtype [  { DEFAULT  | :=  | =  } expression  ]}  [, ...]  ] ) ]
   [ RETURNS rettype [ DETERMINISTIC  ]  | RETURNS TABLE (  { column_name column_type  }
[, ...] )]
   LANGUAGE lang_name
   [
     {IMMUTABLE  | STABLE  | VOLATILE }
      | {SHIPPABLE | NOT SHIPPABLE}
      | WINDOW
      | [ NOT  ] LEAKPROOF
      | {CALLED ON NULL INPUT  | RETURNS NULL ON NULL INPUT | STRICT }
      | {[ EXTERNAL  ] SECURITY INVOKER | [ EXTERNAL  ] SECURITY DEFINER | AUTHID DEFINER  |
AUTHID CURRENT_USER}
      | {fenced | not fenced}
      | {PACKAGE}
      | COST execution_cost
      | ROWS result_rows
      | SET configuration_parameter { {TO | =} value | FROM CURRENT }}
    ][...]
   {
```



```
    AS 'definition'
};
```

- O-compatible syntax for creating a user-defined function:

```
CREATE [ OR REPLACE  ] FUNCTION function_name
   ( [  { argname [ argmode  ] argtype [  { DEFAULT | := | =  } expression  ] }  [, ...]  ] )
   RETURN rettype [ DETERMINISTIC  ]
   [
      {IMMUTABLE  | STABLE  | VOLATILE }
      | {SHIPPABLE | NOT SHIPPABLE}
      | {PACKAGE}
      | {FENCED | NOT FENCED}
      | [ NOT  ] LEAKPROOF
      | {CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
      | {[ EXTERNAL  ] SECURITY INVOKER  | [ EXTERNAL  ] SECURITY DEFINER | AUTHID DEFINER |
AUTHID CURRENT_USER}
      | COST execution_cost
      | ROWS result_rows
      | SET configuration_parameter { {TO | =} value  | FROM CURRENT }
      | LANGUAGE lang_name
   ][...]
   {
     IS  | AS
   } plsql_body
/
```

Parameters

- **function_name**

  Specifies the name of the function to be created (optionally schema-qualified).

  Value range: a string. It must comply with the **naming convention**. The value can contain a maximum of 63 characters. If the value contains more than 63 characters, the database truncates it and retains the first 63 characters as the function name.

  NOTE

  You are advised not to create a function with the same name as a system function. Otherwise, you need to specify the schema of the function when calling the function.

- **argname**

  Specifies the parameter name of the function.

  Value range: a string. It must comply with the **naming convention**. The value can contain a maximum of 63 characters. If the value contains more than 63 characters, the database truncates it and retains the first 63 characters as the function parameter name.

- **argmode**

  Specifies the parameter mode of the function.

  Value range: **IN**, **OUT**, **INOUT**, and **VARIADIC**. The default value is **IN**. Only the parameters of **OUT** can be followed by **VARIADIC**. The parameters of **OUT** and **INOUT** cannot be used in the function definition of RETURNS TABLE.

  NOTE

  VARIADIC specifies parameters of the array type.

- **argtype**

Specifies the data type of a function parameter. **%TYPE** or **%ROWTYPE** can be used to indirectly reference a variable or table type. For details, see **Variable Definition Statements**.

- **expression**

  Specifies the default expression of a parameter.

  NOTE

  - If **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s2**, the default expression is not supported when the parameter is in INOUT mode.
  - It is recommended that you define all default parameters after all non-default parameters.
  - If a function with default parameters is called, input parameters are added to the function from left to right. If inputs of non-default parameters are missing, an error is reported.
  - If **proc_uncheck_default_param** is enabled and a function with default parameters is called, input parameters are added to the function from left to right. The number of defaulted inputs depends on the number of default parameters. If an input of a non-default parameter is missing, the previous default value is used to fill this parameter.
  - When **a_format_version** is set to **10c**, **a_format_dev_version** is set to **s1**, **proc_outparam_override** is disabled, and the function parameters include the output parameter **out** and **default**, the default value cannot be used.

- **rettype**

  Specifies the return data type. Same as **argtype**, **%TYPE** or **%ROWTYPE** can also be used to indirectly reference types.

  When there is **OUT** or **INOUT** parameter, the RETURNS clause can be omitted. If the clause is not omitted, the result type of the clause must be the same as that of the output parameter. If there are multiple output parameters, the result type of the clause is RECORD. Otherwise, the result type of the clause is the same as that of a single output parameter.

  The SETOF modifier indicates that the function will return a set of items, rather than a single item.

  NOTE

  In **FUNCTION argtype** and **rettype** outside **PACKAGE**, **%TYPE** cannot reference the type of the *PACKAGE* variable.

- **column_name**

  Specifies the column name.

- **column_type**

  Specifies the column type.

- **definition**

  Specifies a string constant defining a function. Its meaning depends on the language. It can be an internal function name, a path pointing to a target file, an SQL query, or text in a procedural language.

- **DETERMINISTIC**

  Specifies an API compatible with the SQL syntax. You are advised not to use it.

- **LANGUAGE lang_name**

Specifies the name of the language that is used to implement the function. It can be **SQL**, **internal**, or the name of a customized process language. To ensure downward compatibility, the name can use single quotation marks. Contents in single quotation marks must be capitalized.

Due to compatibility issues, no matter which language is specified when an A-style database is created, the language used is plpgsql.

NOTE

- When an internal function is defined, if AS specifies the function as an internal system function, the parameter type, number of parameters, and return value type of the new function must be the same as those of the internal system function, and the user who creates the internal function must have the permission to execute the internal system function.
- Only users with the sysadmin permission can create internal functions.

- **WINDOW**

  Indicates that this function is a window function. The **WINDOW** attribute cannot be changed when replacing an existing function definition.

NOTICE

For a customized window function, the value of **LANGUAGE** can only be **internal**, and the referenced internal function must be a window function.

- **IMMUTABLE**

  Specifies that the function always returns the same result if the parameter values are the same.

- **STABLE**

  Specifies that the function cannot modify the database, and that within a single table scan it will consistently return the same result for the same parameter value, but its result varies by SQL statements.

- **VOLATILE**

  Specifies that the function value can change in a single table scan and no optimization is performed.

- **SHIPPABLE|NOT SHIPPABLE**

  Specifies whether the function can be pushed down for execution. This port is reserved and is not recommended.

- **FENCED|NOT FENCED**

  Specifies whether the user-defined C function is executed in fenced or not-fenced mode. This port is reserved and is not recommended.

- **PACKAGE**

  Specifies whether the function can be overloaded. PostgreSQL-style functions can be overloaded, and this parameter is designed for functions of other styles.

  – All PACKAGE and non-PACKAGE functions cannot be overloaded or replaced.

  – PACKAGE functions do not support parameters of the VARIADIC type.

  - The **PACKAGE** attribute of functions cannot be modified.
- **LEAKPROOF**

  Specifies that the function has no side effects. **LEAKPROOF** can be set only by the system administrator.
- **CALLED ON NULL INPUT**

  Declares that some parameters of the function can be called in normal mode if the parameter values are null. This parameter can be omitted.
- **RETURNS NULL ON NULL INPUT**

  STRICT

  Specifies that the function always returns null whenever any of its parameters is null. If this parameter is specified, the function is not executed when there are null parameters; instead a null result is returned automatically.

  RETURNS NULL ON NULL INPUT and **STRICT** have the same functions.
- **EXTERNAL**

  The purpose is to be compatible with SQL statements and it is optional. This feature applies to all functions, not only external functions.
- **SECURITY INVOKER**

  AUTHID CURRENT_USER

  Specifies that the function will be executed with the permissions of the user who calls it. This parameter can be omitted.

  SECURITY INVOKER and **AUTHID CURRENT_USER** have the same functions.
- **SECURITY DEFINER**

  AUTHID DEFINER

  Specifies that the function will be executed with the permissions of the user who created it.

  AUTHID DEFINER and **SECURITY DEFINER** have the same functions.
- **COST execution_cost**

  Estimates the execution cost of a function.

  The unit of **execution_cost** is **cpu_operator_cost**.

  Value range: ≥ 0
- **ROWS result_rows**

  Estimates the number of rows returned by the function. This is only allowed when the function is declared to return a set.

  Value range: ≥ 0. The default value is **1000**.
- **configuration_parameter**
  - **value**

    Sets a specified database session parameter to a specified value. If the value is **DEFAULT** or **RESET**, the default setting is used in the new session. **OFF** disables the setting.

    Value range: a string

    - **DEFAULT**
    - **OFF**

  - **RESET**

    Specifies the default value.

  - **FROM CURRENT**

    Uses the value of **configuration_parameter** of the current session.

- **plsql_body**

  Specifies the PL/SQL stored procedure body.

NOTICE

When you are performing operations in the function body related to password or key, such as creating a user, changing a password, encrypting or decrypting, the system catalog and log record plaintext information about the password or key. To prevent sensitive information leakage, you are advised not to perform operations on the function body related to sensitive information, such as passwords or keys.

Examples

```
-- Define a function as SQL query.
gaussdb=# CREATE FUNCTION func_add_sql(integer, integer) RETURNS integer
    AS 'select $1 + $2;'
    LANGUAGE SQL
    IMMUTABLE
    RETURNS NULL ON NULL INPUT;

-- Add an integer by parameter name using PL/pgSQL.
gaussdb=# CREATE OR REPLACE FUNCTION func_increment_plsql(i integer) RETURNS integer AS $$
        BEGIN
            RETURN i + 1;
        END;
    $$ LANGUAGE plpgsql;

-- Return the RECORD type.
gaussdb=# CREATE OR REPLACE FUNCTION func_increment_sql(i int, out result_1 bigint, out result_2 bigint)
    RETURNS SETOF RECORD
    AS $$
    BEGIN
        result_1 = i + 1;
        result_2 = i * 10;
        RETURN next;
    END;
    $$LANGUAGE plpgsql;

-- Return a record containing multiple output parameters.
gaussdb=# CREATE FUNCTION func_dup_sql(in int, out f1 int, out f2 text)
    AS $$ SELECT $1, CAST($1 AS text) || ' is text' $$
    LANGUAGE SQL;

-- Call the func_dup_sql function.
gaussdb=# SELECT * FROM func_dup_sql(42);
 f1 |     f2
----+------------
 42 | 42 is text
(1 row)

-- Compute the sum of two integers and returning the result (if the input is null, the returned result is null):
gaussdb=# CREATE FUNCTION func_add_sql2(num1 integer, num2 integer) RETURN integer
    AS
    BEGIN
```



```
    RETURN num1 + num2;
END;
/


-- Delete the function.
gaussdb=# DROP FUNCTION func_add_sql;
gaussdb=# DROP FUNCTION func_increment_plsql;
gaussdb=# DROP FUNCTION func_increment_sql;
gaussdb=# DROP FUNCTION func_dup_sql;
gaussdb=# DROP FUNCTION func_add_sql2;


-- Set parameters.
gaussdb=# SET behavior_compat_options='proc_outparam_override';

-- Create a function.
gaussdb=# CREATE OR REPLACE FUNCTION func1(in a integer, out b integer)
    RETURNS int
    AS $$
    DECLARE
        c int;
        BEGIN
            c := 1;
            b := a + c;
            RETURN c;
        END; $$
    LANGUAGE 'plpgsql' NOT FENCED;

-- Return return and output parameters at the same time.
gaussdb=# DECLARE
        result integer;
        a integer := 2;
        b integer := NULL;
    BEGIN
        result := func1(a => a, b => b);
        raise info 'b is: %', b;
        raise info 'result is: %', result;
    END;
    /
INFO:  b is: 3
INFO:  result is: 1
ANONYMOUS BLOCK EXECUTE

-- Left assignment expressions are not supported.
gaussdb=# DECLARE
        result integer;
        a integer := 2;
        b integer := NULL;
    BEGIN
        result := func1(a => a, b => b) + 1;
        raise info 'b is: %', b;
        raise info 'result is: %', result;
    END;
    /
ERROR:  when invoking function func1, maybe input something superfluous.
CONTEXT:  compilation of PL/SQL function "inline_code_block" near line 3

-- out/inout in a stored procedure cannot be set to a constant.
gaussdb=# DECLARE
        result integer;
        a integer := 2;
        b integer := NULL;
    BEGIN
        result := func1(a => a, b => 10);
        raise info 'b is: %', b;
        raise info 'result is: %', result;
    END;
    /
```



```
ERROR:  when invoking function func1, no destination for argments "b"
CONTEXT:  compilation of PL/SQL function "inline_code_block" near line 3

-- out/inout in stored procedures can be set to a variable.
gaussdb=# DECLARE
        result integer;
        a integer := 2;
        b integer := NULL;
    BEGIN
        result := func1(a,b);
        raise info 'b is: %', b;
        raise info 'result is: %', result;
    END;
    /
INFO:  b is: 3
INFO:  result is: 1
ANONYMOUS BLOCK EXECUTE

-- Delete the func1 function.
gaussdb=# DROP FUNCTION func1;

-- If the parameter set behavior_compat_options is not set to 'proc_outparam_override', the OUT and IN
OUT output parameters of the function directly called by an anonymous block or stored procedure cannot
be of the composite type, and the return value is used as the first value of the OUT output parameter. As a
result, the invoking fails.
gaussdb=# CREATE TYPE rec as(c1 int, c2 int);
gaussdb=# CREATE OR REPLACE FUNCTION func(a in out rec, b in out int) RETURN int
    AS
    BEGIN
        a.c1:=100;
        a.c2:=200;
        b:=300;
        return 1;
    END;
    /
gaussdb=# DECLARE
        r rec;
        b int;
    BEGIN
  func(r,b);        -- Not supported
    END;
    /
ERROR:  cannot assign non-composite value to a row variable
CONTEXT:  PL/SQL function inline_code_block line 4 at SQL statement
gaussdb=# DROP FUNCTION func;
gaussdb=# DROP TYPE rec;

-- The following examples can be executed only in an A-compatible database:
gaussdb=# CREATE OR REPLACE PACKAGE pkg_type AS
type table_of_index_int is table of integer index by integer; -- Create an integer type.
type table_of_index_int01 is table of table_of_index_int index by integer; -- Create a nested integer type.
type table_of_index_var is table of integer index by varchar(5); -- Create a varchar type.
type table_of_index_var01 is table of table_of_index_var index by varchar(5); -- Create a nested varchar type.
    END pkg_type;
    /

-- Create a function that returns results of the table of integer index by integer type.
gaussdb=# CREATE OR REPLACE FUNCTION func_001(a in out pkg_type.table_of_index_int, b in out
pkg_type.table_of_index_var) --#add in & inout #defult value
    RETURN pkg_type.table_of_index_int
    AS
        table_of_index_int_val pkg_type.table_of_index_int;
        table_of_index_var_val pkg_type.table_of_index_var;
    BEGIN
        for i in 1..2 loop
            table_of_index_int_val(i) := i;
            a(i) := i;
            table_of_index_var_val(i) := i;
            b(i) := i;
```



```
    end loop;
    raise info '%',table_of_index_int_val;
    raise info '%',table_of_index_var_val;
    raise info '%',a;
    raise info '%',b;
    RETURN table_of_index_int_val;
END;
/
gaussdb=# DECLARE
    table_of_index_int_val pkg_type.table_of_index_int;
    table_of_index_var_val pkg_type.table_of_index_var;
begin
    func_001(table_of_index_int_val,table_of_index_var_val);
end;
/
INFO:  {indexbyType:int,1=>1,2=>2}
CONTEXT:  PL/SQL function inline_code_block line 5 at SQL statement
INFO:  {indexbyType:varchar,"1"=>1,"2"=>2}
CONTEXT:  PL/SQL function inline_code_block line 5 at SQL statement
INFO:  {indexbyType:int,1=>1,2=>2}
CONTEXT:  PL/SQL function inline_code_block line 5 at SQL statement
INFO:  {indexbyType:varchar,"1"=>1,"2"=>2}
CONTEXT:  PL/SQL function inline_code_block line 5 at SQL statement
ERROR:  expression is of wrong type
CONTEXT:  PL/SQL function inline_code_block line 5 at SQL statement

-- Create a function containing IN/OUT parameters.
gaussdb=# CREATE OR REPLACE FUNCTION func_001(a in out date, b in out date) --#add in & inout
#defult value
    RETURN integer
    AS
    BEGIN
        raise info '%', a;
        raise info '%', b;
        RETURN 1;
    END;
    /
gaussdb=# DECLARE
    date1 date := '2022-02-02';
    date2 date := '2022-02-02';
BEGIN
    func_001(date1, date2);
END;
/
INFO:  2022-02-02 00:00:00
CONTEXT:  PL/SQL function inline_code_block line 5 at SQL statement
INFO:  2022-02-02 00:00:00
CONTEXT:  PL/SQL function inline_code_block line 5 at SQL statement
ERROR:  invalid input syntax for type timestamp: "1"
CONTEXT:  PL/SQL function inline_code_block line 5 at SQL statement

-- Create a function containing IN/OUT parameters.
gaussdb=# CREATE OR REPLACE FUNCTION func_001(a in out INT, b in out date) --#add in & inout #defult
value
    RETURN INT
    AS
    BEGIN
        raise info '%', a;
        raise info '%', b;
        RETURN a;
    END;
    /
gaussdb=# DECLARE
    date1 int := 1;
    date2 date := '2022-02-02';
BEGIN
    func_001(date1, date2);
END;
/
```



```
INFO:  1
CONTEXT:  PL/SQL function inline_code_block line 5 at SQL statement
INFO:  2022-02-02 00:00:00
CONTEXT:  PL/SQL function inline_code_block line 5 at SQL statement
ANONYMOUS BLOCK EXECUTE

-- Delete the function.
gaussdb=# DROP FUNCTION func_001;

-- Delete the package.
gaussdb=# DROP PACKAGE pkg_type;
```

Helpful Links

ALTER FUNCTION and **DROP FUNCTION**

7.12.8.23 CREATE GLOBAL INDEX

Description

Creates a global secondary index (GSI) on a specified table.

The GSI allows users to define indexes that are inconsistent with the distribution of base tables. In this way, single-node plans for querying non-distributed columns of base tables and unique/primary key constraints on non-distributed columns of base tables are achieved.

NOTE

The centralized system does not support this function.

7.12.8.24 CREATE GROUP

Description

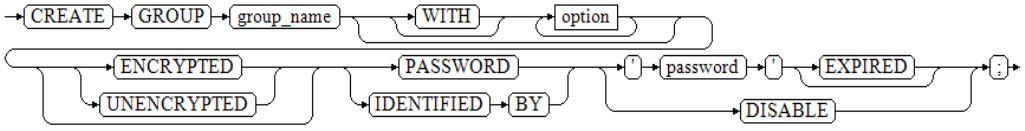
Creates a user group.

Precautions

CREATE GROUP is an alias for CREATE ROLE, and it is not a standard SQL syntax and not recommended. Users can use CREATE ROLE directly.

Syntax

```
CREATE GROUP group_name [ [ WITH ] option [ ... ] ] [ ENCRYPTED | UNENCRYPTED ] { PASSWORD |
IDENTIFIED BY } { 'password' [ EXPIRED ] | DISABLE };
```



The syntax of the option clause (optional) is as follows:

```
{SYSADMIN | NOSYSADMIN}
   | {MONADMIN | NOMONADMIN}
   | {OPRADMIN | NOOPRADMIN}
   | {POLADMIN | NOPOLADMIN}
```



```
  | {AUDITADMIN | NOAUDITADMIN}
  | {CREATEDB | NOCREATEDB}
  | {USEFT | NOUSEFT}
  | {CREATEROLE | NOCREATEROLE}
  | {INHERIT | NOINHERIT}
  | {LOGIN | NOLOGIN}
  | {REPLICATION | NOREPLICATION}
  | {PERSISTENCE | NOPERSISTENCE}
  | CONNECTION LIMIT connlimit
  | VALID BEGIN 'timestamp'
  | VALID UNTIL 'timestamp'
  | RESOURCE POOL 'respool'
  | USER GROUP 'groupuser'
  | PERM SPACE 'spacelimit'
  | TEMP SPACE 'tmpspacelimit'
  | SPILL SPACE 'spillspacelimit'
  | NODE GROUP logic_group_name
  | IN ROLE role_name [, ...]
  | IN GROUP role_name [, ...]
  | ROLE role_name [, ...]
  | ADMIN role_name [, ...]
  | USER role_name [, ...]
  | SYSID uid
  | DEFAULT TABLESPACE tablespace_name
  | PROFILE DEFAULT
  | PROFILE profile_name
  | PGUSER
```

Parameters

See **Parameters** in section "CREATE ROLE."

Examples

```
-- Create a user group. The effect is the same as that of CREATE ROLE.
gaussdb=# CREATE GROUP test_group WITH PASSWORD "********";

-- Run CREATE ROLE to create a role. You cannot log in to the database by default.
-- Run ALTER ROLE role_name WITH LOGIN to enable users to log in to the database.
gaussdb=# CREATE ROLE test_role WITH PASSWORD "********";

-- Run CREATE USER to create a user. A schema with the same name is automatically created and the user has the login permission.
gaussdb=# CREATE USER test_user WITH PASSWORD "********";

-- View the user information.
gaussdb=# \du test*
            List of roles
 Role name  |  Attributes  | Member of
------------+--------------+-----------
 test_group | Cannot login | {}
 test_role  | Cannot login | {}
 test_user  |              | {}

-- Query the schema automatically created by running the CREATE USER command.
gaussdb=# \dn test*
    List of schemas
  Name    |   Owner
----------+-----------
 test_user | test_user
(1 row)

-- Delete.
gaussdb=# DROP ROLE test_role;
gaussdb=# DROP GROUP test_group;
gaussdb=# DROP USER test_user;
```


Helpful Links

ALTER GROUP, **DROP GROUP**, and **CREATE ROLE**

7.12.8.25 CREATE INCREMENTAL MATERIALIZED VIEW

Description

CREATE INCREMENTAL MATERIALIZED VIEW creates a fast-refresh materialized view, and you can refresh the data of the materialized view by using REFRESH MATERIALIZED VIEW (complete-refresh) and REFRESH INCREMENTAL MATERIALIZED VIEW (fast-refresh).

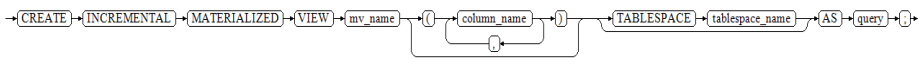
CREATE INCREMENTAL MATERIALIZED VIEW is similar to CREATE TABLE AS, but it remembers the query used to initialize the view, so it can refresh data later. A materialized view has many attributes that are the same as those of a table, but does not support temporary materialized views.

Precautions

- Incremental materialized views cannot be created on database link tables, temporary tables, or global temporary tables.
- Fast-refresh materialized views support only simple filter queries and UNION ALL queries of base tables.
- Distribution columns cannot be specified when an incremental MV is created.
- After a fast-refresh materialized view is created, most DDL operations in the base table are no longer supported.
- IUD operations cannot be performed on fast-refresh materialized views.
- After a fast-refresh materialized view is created, you need to run the **REFRESH** command to synchronize the materialized view with the base table when the base table data changes.
- The Ustore engine does not support the creation and use of materialized views.

Syntax

```
CREATE INCREMENTAL MATERIALIZED VIEW mv_name
    [ (column_name [, ...] ) ]
    [ TABLESPACE tablespace_name ]
    AS query;
```



Parameters

- **mv_name**

  Name (optionally schema-qualified) of the materialized view to be created.

  Value range: a string. It must comply with the **naming convention**.

- **column_name**

  Column name in the new materialized view. The materialized view supports specified columns. The number of specified columns must be the same as that

of columns in the result of the subsequent query statement. If no column name is provided, the column name is obtained from the output column name of the query.

Value range: a string. It must comply with the **naming convention**.

- **TABLESPACE tablespace_name**

  (Optional) Tablespace to which the new materialized view belongs. If not specified, the default tablespace is used.

- **AS query**

  SELECT or **TABLE** command. This query will be run in a security-constrained operation.

Examples

```
-- Create a tablespace.
gaussdb=# CREATE TABLESPACE tbs_data1 RELATIVE LOCATION 'tablespace1/tbs_data1';

-- Create a table.
gaussdb=# CREATE TABLE my_table (c1 int, c2 int) WITH (STORAGE_TYPE = ASTORE);

-- Create a fast-refresh materialized view.
gaussdb=# CREATE INCREMENTAL MATERIALIZED VIEW my_imv (col1,col2) TABLESPACE tbs_data1 AS
SELECT * FROM my_table;

-- Write data to the base table.
gaussdb=# INSERT INTO my_table VALUES(1,1);

-- Query fast-refresh materialized view data.
gaussdb=# SELECT * FROM my_imv;
 col1 | col2
------+------
(0 rows)

-- Fast refresh the fast-refresh materialized view my_imv.
gaussdb=# REFRESH INCREMENTAL MATERIALIZED VIEW my_imv;

-- Query fast-refresh materialized view data.
gaussdb=# SELECT * FROM my_imv;
 col1 | col2
------+------
    1 |    1
(1 row)

-- Delete a fast-refresh materialized view.
gaussdb=# DROP MATERIALIZED VIEW my_imv;

-- Delete my_table.
gaussdb=# DROP TABLE my_table;

-- Drop the tablespace.
gaussdb=# DROP TABLESPACE tbs_data1;
```

Helpful Links

ALTER MATERIALIZED VIEW, **CREATE MATERIALIZED VIEW**, **CREATE TABLE**, **DROP MATERIALIZED VIEW**, **REFRESH INCREMENTAL MATERIALIZED VIEW**, and **REFRESH MATERIALIZED VIEW**

7.12.8.26 CREATE INDEX

Description

Defines a new index.

Indexes are primarily used to enhance database performance (though inappropriate use can result in database performance deterioration). You are advised to create indexes on:

- Columns that are often queried.
- Join conditions. For a query on joined columns, you are advised to create a composite index on the columns. For example, for **select * from t1 join t2 on t1.a=t2.a and t1.b=t2.b**, you can create a composite index on columns **a** and **b** in table **t1**.
- Columns having filter criteria (especially scope criteria) of a WHERE clause.
- Columns that appear after ORDER BY, GROUP BY, and DISTINCT

Partitioned tables do not support partial index creation (when indexes contain the GLOBAL or LOCAL keyword or the created index is a GLOBAL index). If the GLOBAL or LOCAL keyword is declared during index creation, the index of the corresponding type is created. Otherwise, if the partition name is specified during index creation, the LOCAL index is created. If a unique index contains non-partition keys, a GLOBAL index is created. If a unique index contains all partition keys, a LOCAL index is created. Otherwise, a GLOBAL index is created by default.

Precautions

- Indexes consume storage and computing resources. Creating too many indexes has negative impact on database performance (especially the performance of data import. Therefore, you are advised to import the data before creating indexes). Therefore, create indexes only when they are necessary.
- All functions and operators used in an index definition must be immutable, that is, their results must depend only on their parameters and never on any outside influence (such as the contents of another table or the current time). This restriction ensures that the behavior of the index is well-defined. To use a user-defined function in an index or WHERE clause, mark it as an immutable function.
- Partitioned table indexes are classified into local indexes and global indexes. A local index binds to a specific partition, and a global index corresponds to the entire partitioned table.
- A user granted with the CREATE ANY INDEX permission can create indexes in both the public and user schemas.
- If a user-defined function is called in the expression index, the expression index function is executed based on the permission of the function creator.
- Data of the XML type cannot be used as common indexes, unique indexes, global indexes, local indexes, or partial indexes.
- Only B-tree and UB-tree indexes can be created online. Only common indexes of non-partitioned tables, as well as global and local indexes of partitioned tables can be created. PCR UB-tree indexes, level-2 partitions, and GSIs are

not supported. Only common indexes, global indexes, and local indexes of Astore and Ustore can be created online in parallel.

Syntax

- Create an index on a table.

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ IF NOT EXISTS ] [ [schema_name.] index_name ]
ON table_name [ USING method ]
    ({ { column_name [ ( length ) ] | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ]
[ NULLS { FIRST | LAST } ] }[, ...] )
    [ INCLUDE ( column_name [, ...] ) ]
    [ WITH ( {storage_parameter = value} [, ... ] ) ]
    [ TABLESPACE tablespace_name ]
    { [ COMMENT 'string' ] [ ... ] }
    [ { VISIBLE | INVISIBLE } ]
    [ WHERE predicate ];
```

- Create an index on a partitioned table.

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ [schema_name.] index_name ] ON table_name
[ USING method ]
    ( { { column_name [ ( length ) ] | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ]
[ NULLS LAST ] } [, ...] )
    [ LOCAL
        [ ( { PARTITION index_partition_name [ FOR { partition_name | ( partition_value [, ...] ) } ]
[ TABLESPACE index_partition_tablespace ]
                [ ( [SUBPARTITION index_subpartition_name] [ FOR { partition_name | ( partition_value
[, ...] ) } ]  [ TABLESPACE index_partition_tablespace ]
                    [, ...] ) ]
            [, ...] } ) ]
      | GLOBAL ]
    [ INCLUDE ( column_name [, ...] ) ]
    [ WITH ( { storage_parameter = value } [, ...] ) ]
    [ TABLESPACE tablespace_name ]
    { [ COMMENT 'string' ] [ ... ] }
    [ { VISIBLE | INVISIBLE } ];
```

Parameters

- **UNIQUE**

  Creates a unique index. In this way, the system checks whether new values are unique in the index column. Attempts to insert or update data which would result in duplicate entries will generate an error.

  Currently, only B-tree indexes and UB-tree indexes support unique indexes.

- **CONCURRENTLY**

  Creates an index (with ShareUpdateExclusiveLock) in a mode that does not block DML statements. When an index is created, other statements cannot access the table on which the index depends. If this keyword is specified, DML is not blocked during the creation.

  – This option can only specify a name of one index.

  – The CREATE INDEX statement can be run within a transaction, but CREATE INDEX CONCURRENTLY cannot.

  – For temporary tables, you can use CONCURRENTLY to create indexes. However, indexes are created in blocking mode because no other sessions concurrently access the temporary tables and the blocking mode is more cost-effective.

NOTE

- This keyword is specified when an index is created. Astore needs to scan the entire table twice to build it. During the first scan, an index is created and the read and write operations are not blocked. During the second scan, changes that have occurred since the first scan are merged and updated. Ustore needs to scan the entire table only once. During the scan, the data generated by concurrent DML operations is inserted into the temporary table **index_oid_cctmp**. After the scan is complete, the data in the temporary table is merged to the primary index and the temporary table is deleted, the index is created. You can use the GUC parameter **delete_cctmp_table** to specify whether to delete temporary tables after indexes are created online. The default value of this parameter is **on**, indicating that temporary tables are deleted by default. If this parameter is set to **off**, temporary tables are retained.
- For Astore, the table needs to be scanned and built twice, and all existing transactions that may modify the table must be completed. This means that the creation of the index takes a longer time than normal. In addition, the CPU and I/O consumption also affects other services. Although the Ustore only needs to scan the entire table once to create indexes, the preceding consumption also exists.
- Exceptions may occur during online index creation (for example, manual cancellation, duplicate UNIQUE index key values, insufficient resources, thread startup failure, and lock timeout). As a result, online index creation fails. In this case, indexes in the **not valid** state may be retained to avoid occupying system resources. The system automatically clears the failed indexes. Before clearing the failed indexes, wait until the transaction in the table ends. If a long transaction exists, the system may keep waiting. If you cancel the thread, the failed indexes remain. Particularly, if the user cancels the online index creation thread, the process of clearing the failed indexes is started first. If the user cancels the thread again, the process of clearing the failed indexes is ended, and the failed indexes remain. For critical errors, such as FATAL, PANIC, and database faults, you need to manually clear indexes and temporary tables. If both online index creation and automatic residual clearing fail, indexes that are not valid are retained. The residual indexes may be ready or not ready (depending on the phase in which the online index creation fails. For example, the index failed in the first phase is not ready, and the index failed in the third phase is ready). If the residual indexes are ready, the DML statements still maintain the residual indexes. During the maintenance, other errors may occur (for example, the size of the index column exceeds the maximum value or the unique index constraint is violated). To prevent residual indexes from occupying system resources and generating unexpected errors, you need to manually delete them as soon as possible.
- For Astore, after the second scan, index creation must wait for any transaction that holds a snapshot earlier than the snapshot taken by the second scan to end. In addition, the ShareUpdateExclusiveLock (level 4) added during index creation conflicts with a lock whose level is greater than or equal to 4. Therefore, when such an index is created, the system is prone to hang or deadlock. Example:
  - If two sessions create an index by using CONCURRENTLY for the same table, a deadlock occurs.
  - If a session creates an index by using CONCURRENTLY for a table and the other session drops a table, a deadlock occurs.
  - There are three sessions. Session 1 locks table **a** and does not commit it. Session 2 creates an index by using CONCURRENTLY for table **b**. Session 3 writes data to table **a**. Before the transaction of session 1 is committed, session 2 is blocked.
  - The transaction isolation level is set to repeatable read (read committed by default). Two sessions are started. Session 1 writes data to table **a** and does not commit it. Session 2 creates an index by using CONCURRENTLY for table **b**. Before the transaction of session 1 is committed, session 2 is blocked.
- When the I/O and CPU resources are not limited, the service performance deterioration caused by online index creation can be controlled within 10%.

However, in special scenarios, the service performance deterioration may exceed 10%. This is because online index creation is a long transaction that consumes a large number of I/O and CPU resources. It consumes more resources than offline index creation. The longer the online index creation transaction lasts, the greater the impact on service performance. The time for creating indexes online is positively correlated with the data volume of base tables and the data volume generated by concurrent DML statements. When the I/O and CPU resources are not limited, the time for creating indexes online is about two to six times that for creating indexes offline. However, when the number of concurrent transactions is large (> 10000 TPS) or resource contention occurs, the time may be even longer. If a long transaction exists during online index creation, the long transaction running time must be added. In Astore and Ustore modes, you can create indexes in parallel to shorten the index creation time. The performance of online parallel index creation increases to a certain value and becomes stable as the number of parallel worker threads increases. Compared with the performance of creating indexes in serial mode, the performance of creating indexes in parallel online is improved by about 30%. You are advised to create indexes online during off-peak hours to avoid great impact on services. In addition, do not execute long transactions during online index creation. Although online index creation provides the capability of uninterrupted services to some extent, it still needs to be implemented with caution.

- When a UNIQUE index is created online (by using CREATE UNIQUE INDEX CONCURRENTLY), the table is scanned to check whether the data in the table is unique. If the data in the table is not unique, an error is reported and the online index creation exits. During online index creation, whether data in the table is unique may change. Whether the creation fails depends on whether duplicate data is scanned. Consider the following situations: 1. Tuple A and tuple B are inserted during table scanning, and their index columns are the same. If both tuple A and tuple B are scanned during online index scanning, a uniqueness violation may be reported. However, if tuple B is deleted later, in this case, an error may be reported during online index creation, indicating that the uniqueness is violated. However, after the online index creation fails and exits, the data in the table is found to be unique. 2. During online index creation, if tuple A is inserted, deleted, and then inserted, and SnapshotNow is used to scan the table, tuple A may be scanned twice, which may violate the uniqueness constraint. During online index creation, Astore uses SnapshotMVCC to scan tables, and Ustore uses SnapshotNow to scan tables.

- **IF NOT EXISTS**

  When IF NOT EXISTS is specified, the system checks whether a relationship with the same name already exists in the current schema before creating an index. It is not created and a NOTICE is returned if a relationship with the same name already exists. When IF NOT EXISTS is not specified and a relationship with the same name exists in the schema, an ERROR is returned.

- **schema_name**

  Specifies the schema name.

  Value range: an existing schema name

- **index_name**

  Specifies the name of the index to be created. The schema of the index is the same as that of the table.

  Value range: a string. It must comply with the **naming convention**.

- **table_name**

  Specifies the name of the table to be indexed (optionally schema-qualified).

  Value range: an existing table name

- **USING method**

Specifies the name of the index method to be used.

Value range:

- **btree**: B-tree indexes store key values of data in a B+ tree structure. This structure helps users to quickly search for indexes. B-tree is applicable to comparison query and range query.
- **ubtree**: Multi-version B-tree index used only for Ustore tables. The index page contains transaction information and can be recycled. By default, the INSERTPT function is enabled for UB-tree indexes.

Row-store tables (Astore storage engine) support the following index types: **btree** (default). Row-store tables (Ustore storage engine) support the following index type: **ubtree**.

NOTICE

B-tree and UB-tree are closely related to the table storage type Astore or Ustore. When an index is created, if the specified index type does not correspond to the main table, the index type is automatically converted.

- **column_name**

  Specifies the name of the column on which an index is to be created.

  Multiple columns can be specified if the index method supports multi-column indexes. A global index supports a maximum of 31 columns, and other indexes support a maximum of 32 columns.

- **column_name (length)**

  Creates a prefix key index based on a column in the table. **column_name** indicates the column name of the prefix key, and **length** indicates the prefix length.

  The prefix key uses the prefix of the specified column data as the index key value, which reduces the storage space occupied by the index. Indexes can be used for partial filter and join conditions that contain prefix key columns.

  NOTE

  - The prefix key supports the following index methods: btree and ubtree.
  - The data type of the prefix key column must be binary or character (excluding special characters).
  - The prefix length must be a positive integer that does not exceed 2676 and cannot exceed the maximum length of the column. For the binary type, the prefix length is measured in bytes. For non-binary character types, the prefix length is measured in characters. The actual length of the key value is restricted by the internal page. If a column contains multi-byte characters or an index has multiple keys, the length of the index line may exceed the upper limit. As a result, an error is reported. Consider this situation when setting a long prefix length.
  - In the CREATE INDEX syntax, the following keywords cannot be used as prefix keys for column names: COALESCE, EXTRACT, GREATEST, LEAST, LNNVL, NULLIF, NVL, NVL2, OVERLAY, POSITION, REGEXP_LIKE, SUBSTRING, TIMESTAMPDIFF, TREAT, TRIM, XMLCONCAT, XMLELEMENT, XMLEXISTS, XMLFOREST, XMLPARSE, XMLPI, XMLROOT and XMLSERIALIZE.
  - The prefix key is a special expression key. Some constraints and restrictions that are not described are the same as those of the expression key. For details, see the description of the expression index.

- **expression**

  Specifies an expression based on one or more columns of the table. The expression usually must be written with surrounding parentheses, as shown in the syntax. However, the parentheses can be omitted if the expression has the form of a function call.

  Expression can be used to obtain fast access to data based on some transformation of the basic data. For example, an index computed on upper(col) would allow the clause WHERE upper(col) = 'JIM' to use an index.

  If an expression contains IS NULL, the index for this expression is invalid. In this case, you are advised to create a partial index.

- **COLLATE collation**

  Assigns a collation to the column (which must be of a collatable data type). If no collation is specified, the default collation is used. You can run the **SELECT * FROM pg_collation** command to query collation rules from the pg_collation system catalog. The default collation rule is the row starting with **default** in the query result.

- **opclass**

  Specifies the name of an operator class. An operator class can be specified for each column of an index. The operator class identifies the operators to be used by the index for that column. For example, a B-tree index on the type int4 would use int4_ops; this operator class includes comparison functions for values of type int4. In practice, the default operator class for the column's data type is sufficient. The operator class applies to data with multiple sorts. For example, users might want to sort a complex-number data type either by absolute value or by real part. They could do this by defining two operator classes for the data type and then selecting the proper class when making an index.

- **ASC**

  Specifies an ascending (default) sort order.

- **DESC**

  Specifies a descending sort order.

- **NULLS FIRST**

  Specifies that null values appear before non-null values in the sort ordering. This is the default when **DESC** is specified.

- **NULLS LAST**

  Specifies that null values appear after non-null values in the sort ordering. This is the default when **DESC** is not specified.

- **LOCAL**

  Specifies that the partitioned index to be created is a local index.

- **GLOBAL**

  Specifies the partitioned index to be created as a global index. If no keyword is specified, a global index is created by default.

- **INCLUDE (column_name [, ...]**)

  The optional **INCLUDE** clause specifies that some non-key columns are included in indexes. Non-key columns cannot be used as search criteria for accelerating index scans, and they are omitted when the UNIQUE constraints of the indexes are checked.

An index-only scan can directly return content in the non-key columns without accessing the heap table corresponding to the indexes.

Exercise caution when adding non-key columns as **INCLUDE** columns, especially for wide columns. If the size of an index tuple exceeds the maximum size allowed by the index type, data insertion fails. Note that in any case, adding non-key columns to an index increases the space occupied by the index, which may slow down the search speed.

Currently, only UB-tree indexes access mode supports this feature. Non-key columns are stored in the index leaf tuple corresponding to the heap tuple and are not included in the tuple on the upper-layer index page.

- **PARTITION clause**

  The syntax for specifying level-1 partition information is as follows:

  ```
  PARTITION index_partition_name [ FOR { partition_name | ( partition_value [, ...] ) } ] [ TABLESPACE index_partition_tablespace ]
  ```

  The syntax for specifying level-2 partition information is as follows:

  ```
  SUBPARTITION index_subpartition_name [ FOR { partition_name | ( partition_value [, ...] ) } ] [ TABLESPACE index_partition_tablespace ]
  ```

  If **for partition_name** or **partition_value** is specified in the PARTITION clause, the created partitioned index has the sparsely partitioned index attribute. If **for partition_name** or **partition_value** is not specified, the number of PARTITION clauses must be the same as the number of partitions in the target table.

  NOTE

  - The sparsely partitioned index can be created only on a single partition.
  - The sparsely partitioned index supports only B-tree and UB-tree indexes.
  - The sparsely partitioned index does not support the UNIQUE attribute.
  - The created sparsely partitioned index contains the **sparsely_partitioned=true** option. Note that the CREATE TABLE or ALTER TABLE statement cannot be used to explicitly specify a non-sparsely partitioned index as a sparsely partitioned index in this option. If the created index is a sparsely partitioned index, **sparsely_partitioned=false** will be ignored.
  - If a version earlier than 505.0.0 is upgraded to 505.0.0 or later, sparsely partitioned indexes cannot be created during upgrade observation.
  - If the specified partition does not exist during the sparsely partitioned index creation for partition_value and the partitioned table supports automatic partitioning, a new partition is created and the sparsely partitioned index is created in the new partition. The autonomous transaction solution is used to create a partition. After the primary transaction is rolled back due to an exception, the new partition may still exist.

  – **PARTITION index_partition_name**

    Specifies the name of an index partition.

    Value range: a string. It must comply with the **naming convention**.

  – **SUBPARTITION index_subpartition_name**

    Specifies the name of a level-2 index partition.

    Value range: a string. It must comply with the **naming convention**.

  – **FOR partition_name**

    Specifies the name of the target partition. If the specified partition does not exist, an error is reported.

  - **FOR (partition_value [, ...])**

    Specifies the value of the specified partition key. The level-1 partition key values are placed in the partition key value FOR list of the level-1 partition, and the level-2 partition key values are placed in the partition key value FOR list of the level-2 partition. If there are multiple columns of partition keys, specify multiple **partition_value** values.

  - **TABLESPACE index_partition_tablespace**

    Specifies the tablespace of an index partition.

    Value range: If this parameter is not specified, the value of **index_tablespace** is used.

- **WITH ({storage_parameter = value} [, ...])**

  Specifies the storage parameter used for an index.

  Value range:

  For indexes other than Psort, you can also set it to **FILLFACTOR**. Only UB-tree indexes support **INDEXSPLIT**. Only non-partitioned B-tree indexes support the **DEDUPLICATION** parameter. Only UB-tree indexes support the **INDEX_TXNTYPE** parameter.

  - STORAGE_TYPE

    Storage engine type of the table where the index is located. If **storage_type** specified by the index conflicts with the index type, the storage engine type is automatically changed to the correct one. Only B-tree and UB-tree are supported. This parameter can be specified only once.

    Value range: **USTORE** (The table where the index is located uses the in-place update storage engine.) and **ASTORE** (The table where the index is located uses the append-only storage engine.)

    Default value: **USTORE** for Ustore tables and **ASTORE** for Astore tables.

  - FILLFACTOR

    The fill factor of an index is a percentage from 10 to 100. In the scenario where a large number of concurrent insertions are performed and the key value range is dense, the contention of the same index page is high during the insertion. Therefore, a smaller fill factor is more appropriate.

    Value range: 10–100

  - INDEXSPLIT

    Specifies the splitting policy of UB-tree indexes. The **DEFAULT** policy is the same as the splitting policy of B-tree indexes. The **INSERTPT** policy can significantly reduce the index space usage in some scenarios.

    Value range: **INSERTPT** and **DEFAULT**

    Default value: **INSERTPT**

  - INDEX_TXNTYPE

    Specifies the UB-tree index type (only UB-tree indexes support **INDEX_TXNTYPE**). When the value is **PCR**, flashback queries can be performed using UB-tree. PCR UB-tree indexes do not support online index creation, GSIs, ultimate RTO replay, and read on standby. If **INDEX_TXNTYPE** is not specified, the GUC parameter **index_txntype** specifies the type of the index to be created. **INDEX_TXNTYPE** cannot be

modified by using ALTER INDEX INDEX_NAME SET (INDEX_TXNTYPE=PCR or RCR).

Type: character string (case insensitive)

Value range: **RCR** and **PCR**

Default value: **RCR**

For example:

```
CREATE UNIQUE INDEX t2_b_pkey ON t(b) WITH(index_txntype='pcr');
```

- ACTIVE_PAGES

  Specifies the number of index pages, which may be less than the actual number of physical file pages and can be used for optimization. Currently, this parameter is valid only for the local index of the Ustore partitioned table and will be updated by VACUUM and ANALYZE (including AUTOVACUUM). You are advised not to manually set this parameter.

- DEDUPLICATION

  Specifies whether to deduplicate and compress tuples with duplicate key values. This is an index parameter. When there are a large number of indexes with duplicate key values, enabling this parameter can effectively reduce the space occupied by indexes. This parameter does not take effect for primary key indexes and unique indexes. If non-unique indexes are used and the index key value duplication rate is low or the index key values are unique, enabling this parameter will slightly deteriorate the index insertion performance. Currently, local and global indexes of partitioned tables are not supported.

  Value range: Boolean value. The default value is the value of the **enable_default_index_deduplication** GUC parameter (the default value is **off**).

- stat_state

  Determines whether index statistics are locked. If locked, the index statistics cannot be updated.

  Value range: **locked** and **unlock**.

  Default value: **unlock**.

- enable_tde

  Specifies whether the index is encrypted. If the index is encrypted, the database automatically encrypts the data in the encrypted index before storing it. Before using this parameter, ensure that the TDE function has been enabled using the GUC parameter **enable_tde**, the information for accessing the key service has been set using the GUC parameter **tde_key_info**, and the **enable_tde** attribute has been set for the base table. For details about how to use this parameter, see section "Transparent Data Encryption" in *Feature Guide*. This parameter supports only B-tree and UB-tree indexes. Other indexes such as hash indexes are not supported.

  Value range: **on** and **off** If **enable_tde** is set to **on**, the values of **key_type**, **tde_cmk_id**, and **dek_cipher** are automatically generated in the database.

  Default value: **off**

- encrypt_algo

Specifies the algorithm for encrypting indexes. This parameter must be used together with **enable_tde**.

Value range: a string. The value can be **AES_128_CTR** or **SM4_CTR**.

Default value: null if **enable_tde** is not set, or **AES_128_CTR** if **enable_tde** is set.

- dek_cipher

  Specifies the DEK ciphertext. After you set the **enable_tde** parameter for an index, the index automatically copies the **dek_cipher** parameter in the base table. This parameter cannot be set or modified.

  Value range: a string

  Default value: null

- key_type

  Specifies the type of the master key. After you set the **enable_tde** parameter for a table, the index automatically copies the **key_type** parameter of the base table. This parameter cannot be set or modified.

  Default value: null

- cmk_id

  Specifies the ID of the master key. After you set the **enable_tde** parameter for a table, the index automatically copies the **cmk_id** parameter of the base table. This parameter cannot be set or modified.

  Value range: a string

  Default value: null

- LPI_PARALLEL_METHOD

  Index parameter, which is used to set the parallel creation mode of local indexes for partitioned tables.

  Type: string

  Value range: **PAGE**, **PARTITION**, and **AUTO PAGE** creates indexes for pages in parallel. Multiple subthreads are started to scan and sort data. Each subthread processes a data page at a time. After scanning and sorting, the main thread combines the sorting results in serial mode and inserts tuples into the indexes. **PARTITION** concurrently creates indexes at the partition level and starts multiple subthreads. Each subthread scans, sorts, and inserts indexes into a partition. **AUTO** estimates the cost of parallel index creation at the page level and partition level based on the partitioned table statistics, and selects the parallel index creation mode with a lower cost. (The statistics may be different from the actual data, resulting in inaccurate calculation results.)

  Default value: **PAGE**

  Setting suggestion: Set this parameter to **PARTITION** if the partitioned table data is evenly distributed in each partition. This parameter supports only the B-tree local index of Astore partitioned tables. It does not support the global index of partitioned tables, non-partitioned table indexes, segment-page table indexes, or online index creation.

  For example:

  ```
  CREATE INDEX idx ON tbl(col) WITH (lpi_parallel_method = 'partition');
  ```

- **TABLESPACE tablespace_name**

Specifies the tablespace for an index. If no tablespace is specified, the default tablespace is used.

Value range: an existing table name

- **WHERE predicate**

  Creates a partial index. A partial index is an index that contains entries for only a portion of a table, usually a portion that is more useful for indexing than the rest of the table. For example, if you have a table that contains both billed and unbilled orders where the unbilled orders take up a small fraction of the total table and yet that is an often used portion, you can improve performance by creating an index on just that portion. In addition, WHERE with UNIQUE can be used to enforce uniqueness over a subset for a table.

  Value range: The predicate expression can only refer to columns of the underlying table, but it can use all columns, not just the ones being indexed. Currently, subqueries and aggregate expressions are forbidden in WHERE. You are advised not to use numeric types such as int for predicate, because such types can be implicitly converted to bool values (non-zero values are implicitly converted to **true** and **0** is implicitly converted to **false**), which may cause unexpected results.

  For a partitioned table index, if the created index contains the GLOBAL keyword or the created index is a GLOBAL index, the WHERE clause cannot be used to create an index.

- **COMMENT 'string'**

  COMMENT 'string' is used to add comments to an index.

  NOTICE

  - This clause is valid only when **sql_compatibility** is set to **'B'**.
  - An index-level comment can contain a maximum of 1024 characters.

- **VISIBLE**

  Sets the index to be visible. This is the default option.

  NOTE

  - VISIBLE can be set only in an A-compatible database (that is, **sql_compatibility** set to **'A'**).
  - When **disable_keyword_options** is set to **"visible"**, this keyword cannot be used.
  - This keyword is not supported in the upgrade uncommitted phase.

- **INVISIBLE**

  Sets the index to be invisible.

  NOTE

  - INVISIBLE can be set only in an A-compatible database (that is, **sql_compatibility** set to **'A'**).
  - When **disable_keyword_options** is set to **"invisible"**, this keyword cannot be used.
  - This keyword is not supported in the upgrade uncommitted phase.

Examples

- **Common index**

```
-- Create the tbl_test1 table.
gaussdb=# CREATE TABLE tbl_test1(
    id int,           -- User ID
    name varchar(50), -- Username
    postcode char(6)  -- Postal code
);

-- Create the tbs_index1 tablespace.
gaussdb=# CREATE TABLESPACE tbs_index1 RELATIVE LOCATION 'test_tablespace/tbs_index1';

-- Create the idx_test1 index for the tbl_test1 table and specify a tablespace.
gaussdb=# CREATE INDEX idx_test1 ON tbl_test1(name) TABLESPACE tbs_index1;

-- Query information about the idx_test1 index.
gaussdb=# SELECT indexname,tablename,tablespace FROM pg_indexes WHERE indexname =
'idx_test1';
 indexname | tablename | tablespace
-----------+-----------+------------
 idx_test1 | tbl_test1 | tbs_index1
(1 row)

-- Delete the index.
gaussdb=# DROP INDEX idx_test1;

-- Drop the tablespace.
gaussdb=# DROP TABLESPACE tbs_index1;
```

- **Unique index**

```
-- Create the unique index idx_test2 for the tbl_test1 table.
gaussdb=# CREATE UNIQUE INDEX idx_test2 ON tbl_test1(id);

-- Query index information.
gaussdb=# \d tbl_test1
          Table "public.tbl_test1"
  Column  |         Type          | Modifiers
----------+-----------------------+-----------
 id       | integer               |
 name     | character varying(50) |
 postcode | character(6)          |
Indexes:
    "idx_test2" UNIQUE, btree (id) TABLESPACE pg_default

-- Delete the index.
gaussdb=# DROP INDEX idx_test2;
```

- **Expression index**

```
-- Create an expression index for the tbl_test1 table.
gaussdb=# CREATE INDEX idx_test3 ON tbl_test1(substr(postcode,2));

-- Query index information.
gaussdb=# \d tbl_test1
          Table "public.tbl_test1"
  Column  |         Type          | Modifiers
----------+-----------------------+-----------
 id       | integer               |
 name     | character varying(50) |
 postcode | character(7)          |
Indexes:
    "idx_test3" btree (substr(postcode::text, 2)) TABLESPACE pg_default

-- Delete the index.
gaussdb=# DROP INDEX idx_test3;
```

- **Partial index**

```
-- Create an index for data whose ID is not empty in the tbl_test1 table.
gaussdb=# CREATE INDEX idx_test4 ON tbl_test1(id) WHERE id IS NOT NULL;
```



```
-- Delete the index.
gaussdb=# DROP INDEX idx_test4;

-- Delete the table.
gaussdb=# DROP TABLE tbl_test1;
```

- **Partitioned index**

```
-- Create a table.
gaussdb=# CREATE TABLE student(id int, name varchar(20)) PARTITION BY RANGE (id) (
    PARTITION p1 VALUES LESS THAN (200),
    PARTITION pmax VALUES LESS THAN (MAXVALUE)
);

-- Create a local partitioned index without specifying the index partition name.
gaussdb=# CREATE INDEX idx_student1 ON student(id) LOCAL;

-- Check the index partition information. It is found that the number of local index partitions is the
same as the number of table partitions.
gaussdb=# SELECT relname FROM pg_partition WHERE parentid = 'idx_student1'::regclass;
   relname
-------------
 p1_id_idx
 pmax_id_idx
(2 rows)

-- Delete the local partitioned index.
gaussdb=# DROP INDEX idx_student1;

-- Create a global index.
gaussdb=# CREATE INDEX idx_student2 ON student(name) GLOBAL;

-- Check the index partition information. It is found that the number of global index partitions is
different from the number of table partitions.
gaussdb=# SELECT relname FROM pg_partition WHERE parentid = 'idx_student2'::regclass;
 relname
---------
(0 rows)

-- Delete the global partitioned index.
gaussdb=# DROP INDEX idx_student2;

-- Create a local expression index without specifying the index partition name.
gaussdb=# CREATE INDEX idx_student3 ON student(lower(name)) LOCAL;

-- Check the index partition information. It is found that the number of local index partitions is the
same as the number of table partitions.
gaussdb=# SELECT relname FROM pg_partition WHERE parentid = 'idx_student3'::regclass;
   relname
-------------
 p1_id_idx
 pmax_id_idx
(2 rows)

-- Delete the expression index of the LOCAL partition.
gaussdb=# DROP INDEX idx_student3;

-- Create a global expression index.
gaussdb=# CREATE INDEX idx_student4 ON student(lower(name)) GLOBAL;

-- Check the index partition information. It is found that the number of global expression index
partitions is different from the number of table partitions.
gaussdb=# SELECT relname FROM pg_partition WHERE parentid = 'idx_student4'::regclass;
 relname
---------
(0 rows)

-- Delete the expression index of the global partition.
gaussdb=# DROP INDEX idx_student4;
```



```
-- Delete the table.
gaussdb=# DROP TABLE student;
```

Helpful Links

ALTER INDEX and **DROP INDEX**

Suggestions

- create index

  Constraints:

  - An index of an ordinary table supports a maximum of 32 columns. A GLOBAL index of a partitioned table supports a maximum of 31 columns.
  - The size of a single index cannot exceed the size of the index page (8 KB). The size of a B-tree or UB-tree cannot exceed one-third of the page size.
  - Partial indexes cannot be created in a partitioned table.
  - When a GLOBAL index is created on a partitioned table, the following constraints apply:
    - Expression indexes and partial indexes are not supported.
    - Only B-tree indexes are supported.
  - In the same attribute column, the local index and global index of a partition cannot coexist.
  - If the ALTER statement does not contain UPDATE GLOBAL INDEX, the original global index is invalid. In this case, other indexes are used for query. If the ALTER statement contains UPDATE GLOBAL INDEX, the original global index is still valid and the index function is correct.

7.12.8.27 CREATE LANGUAGE

This version does not support this syntax.

7.12.8.28 CREATE MASKING POLICY

Description

Creates a masking policy.

Precautions

- Only users with the poladmin or sysadmin permission, or the initial user can perform this operation.
- The masking policy takes effect only after the security policy is enabled, that is, **enable_security_policy** is set to **on**.

> **CAUTION**
>
> When you use database links to perform operations on remote objects, the client initiates a database link request. The actual sender is the server, and the attributes such as the IP address of the sender are the values of the server. For details, see **DATABASE LINK**.

Syntax

```
CREATE MASKING POLICY policy_name masking_clause[, ...] [ policy_filter_clause ] [ENABLE | DISABLE];
```

- masking_clause
  ```
  masking_function ON LABEL(label_name[, ...])
  ```
- masking_function

  maskall is not a preset function and cannot be displayed by running **\df**.

  The masking methods during presetting are as follows:
  ```
  { maskall | randommasking | creditcardmasking | basicemailmasking | fullemailmasking |
  shufflemasking | alldigitsmasking | regexpmasking }
  ```
- policy_filter_clause:
  ```
  FILTER ON { FILTER_TYPE ( filter_value [, ...] ) } [, ...]
  ```
- FILTER_TYPE:
  ```
  { APP | ROLES | IP }
  ```

Parameters

- **policy_name**

  Specifies the masking policy name, which must be unique.

  Value range: a string. It must comply with the **naming convention**.
- **label_name**

  Specifies the resource label name.
- **masking_clause**

  Specifies the masking function to be used to anonymize database resources labeled by **label_name**. **schema.function** can be used to specify the masking function.
- **policy_filter**

  Specifies the users for which the masking policy takes effect. If this parameter is left empty, the masking policy takes effect for all users.
- **FILTER_TYPE**

  Specifies the types of information to be filtered by the policy, including **IP**, **APP**, and **ROLES**.
- **filter_value**

  Indicates the detailed information to be filtered, such as the IP address, app name, and username.
- **ENABLE|DISABLE**

  Enables or disables the masking policy. If **ENABLE|DISABLE** is not specified, **ENABLE** is used by default.

Examples

- Create a masking policy using maskall. (All values of the character string type are masked as **x**, and other types are displayed as the default values of this type.)

```
-- Create table tb_for_masking.
gaussdb=# CREATE TABLE tb_for_masking(idx int, col1 text, col2 text, col3 text, col4 text, col5 text,
col6 text, col7 text,col8 text);

-- Insert data into the tb_for_masking table.
gaussdb=# INSERT INTO tb_for_masking VALUES(1, '9876543210', 'usr321usr', 'abc@huawei.com',
'abc@huawei.com', '1234-4567-7890-0123', 'abcdef 123456 ui 323 jsfd321 j3k2l3',
'4880-9898-4545-2525', 'this is a llt case');

-- View data.
gaussdb=# SELECT * FROM tb_for_masking;
 idx |    col1    |   col2    |      col3      |      col4      |       col5         |               col6
     |        col7         |        col8
-----+------------+-----------+----------------+----------------+--------------------
+------------------------------
------+---------------------+--------------------
   1 | 9876543210 | usr321usr | abc@huawei.com | abc@huawei.com | 1234-4567-7890-0123 | abcdef
123456 ui 323 jsfd321 j
3k2l3 | 4880-9898-4545-2525 | this is a llt case
(1 row)

-- Create a resource label for sensitive column col1.
gaussdb=# CREATE RESOURCE LABEL mask_lb1 ADD COLUMN(tb_for_masking.col1);

-- Create a data masking policy named maskpol1.
gaussdb=# CREATE MASKING POLICY maskpol1 maskall ON LABEL(mask_lb1);

-- Access the tb_for_masking table. The masking policy is triggered for the col1 column.
gaussdb=# SELECT col1 FROM tb_for_masking;
    col1
------------
 xxxxxxxxxx
(1 row)
```

- Create a data masking policy that uses random masking to mask values of the character string type to random numbers. The masked values are different each time.

```
-- Create a resource label for sensitive column col2.
gaussdb=# CREATE RESOURCE LABEL mask_lb2 ADD COLUMN(tb_for_masking.col2);

-- Create a data masking policy named maskpol2.
gaussdb=# CREATE MASKING POLICY maskpol2 randommasking ON LABEL(mask_lb2);

-- Access the tb_for_masking table. The masking policy is triggered for the col2 column.
gaussdb=# SELECT col2 FROM tb_for_masking;
   col2
-----------
 0e8612d9a
(1 row)
```

- Create a data masking policy that uses basicemailmasking to set all data content before the at sign (@) in the mailbox format value of the character string type to **x**.

```
-- Create a resource label for sensitive column col3.
gaussdb=# CREATE RESOURCE LABEL mask_lb3 ADD COLUMN(tb_for_masking.col3);

-- Create a data masking policy named maskpol3.
gaussdb=# CREATE MASKING POLICY maskpol3 basicemailmasking ON LABEL(mask_lb3);

-- Access the tb_for_masking table. The masking policy is triggered for the col3 column.
gaussdb=# SELECT col3 FROM tb_for_masking;
     col3
----------------
```



```
 xxx@huawei.com
(1 row)
```

- Create a data masking policy that uses fullemailmasking to retain only the at sign (@) and **.com** in the email address format of the character string type, and set other parameters to **x**.

```
-- Create a resource label for sensitive column col4.
gaussdb=# CREATE RESOURCE LABEL mask_lb4 ADD COLUMN(tb_for_masking.col4);

-- Create a data masking policy named maskpol4.
gaussdb=# CREATE MASKING POLICY maskpol4 fullemailmasking ON LABEL(mask_lb4);

-- Access the tb_for_masking table. The masking policy is triggered for the col4 column.
gaussdb=# SELECT col4 FROM tb_for_masking;
    col4
----------------
 xxx@xxxxxx.com
(1 row)
```

- Create a masking policy that uses creditcardmasking to retain the hyphen (-) and the last four digits for values of the string type, and set the rest to **x**.

```
-- Create a resource label for sensitive column col5.
gaussdb=# CREATE RESOURCE LABEL mask_lb5 ADD COLUMN(tb_for_masking.col5);

-- Create a data masking policy named maskpol5.
gaussdb=# CREATE MASKING POLICY maskpol5 creditcardmasking ON LABEL(mask_lb5);

-- Access the tb_for_masking table. The masking policy is triggered for the col5 column.
gaussdb=# SELECT col5 FROM tb_for_masking;
       col5
---------------------
 xxxx-xxxx-xxxx-0123
(1 row)
```

- Create a data masking policy that uses shuffle masking to shuffle the positions of characters in the value of the character string type.

```
-- Create a resource label for sensitive column col6.
gaussdb=# CREATE RESOURCE LABEL mask_lb6 ADD COLUMN(tb_for_masking.col6);

-- Create a data masking policy named maskpol6.
gaussdb=#  CREATE MASKING POLICY maskpol6 shufflemasking ON LABEL(mask_lb6);

-- Access the tb_for_masking table. The masking policy is triggered for the col6 column.
gaussdb=# SELECT col6 FROM tb_for_masking;
              col6
-------------------------------------
 2 b6jusfd54c3312 13d23lk3jf3  2eai
(1 row)
```

- Create a masking policy that uses regexpmasking to mask values of the string type using regular expressions.

```
-- Create a resource label for sensitive column col7.
gaussdb=# CREATE RESOURCE LABEL mask_lb7 ADD COLUMN(tb_for_masking.col7);

-- Create a data masking policy named maskpol7.
gaussdb=# CREATE MASKING POLICY maskpol7 regexpmasking('[\d+]','*',2, 9) ON LABEL(mask_lb7);

-- Access the tb_for_masking table. The masking policy is triggered for the col7 column.
gaussdb=# SELECT col7 FROM tb_for_masking;
       col7
---------------------
 48**-****-*545-2525
(1 row)
```

- Create a masking policy that takes effect only for scenarios where users are **dev_mask** and **bob_mask**, the client tool is **gsql**, and IP addresses are **172.31.17.160** and **127.0.0.0/24**.


```
-- Create users dev_mask and bob_mask.
gaussdb=# CREATE USER dev_mask PASSWORD '*********';
gaussdb=# CREATE USER bob_mask PASSWORD '*********';

-- Create a resource label for sensitive column col8.
gaussdb=# CREATE RESOURCE LABEL mask_lb8 ADD COLUMN(tb_for_masking.col8);

-- Create a data masking policy named maskpol8.
gaussdb=# CREATE MASKING POLICY maskpol8 randommasking ON LABEL(mask_lb8) FILTER ON
ROLES(dev_mask, bob_mask), APP(gsql), IP('172.31.17.160', '127.0.0.0/24');

-- Access the tb_for_masking table. The masking policy is triggered for the col8 column.
gaussdb=# SELECT * FROM tb_for_masking;

-- Use the gsql tool, set the IP address to 172.31.17.160, and view tb_for_masking as user dev_mask.
gaussdb=# GRANT ALL PRIVILEGES TO dev_mask;

-- Use maskpol8 for data masking. The result is random and different each time.
gaussdb=# SELECT col8 FROM tb_for_masking;
       col8
--------------------
 cf32a9aa427f219ab0
(1 row)

gaussdb=# SELECT col8 FROM tb_for_masking;
       col8
--------------------
 13efa056dda1e1a474
(1 row)
```

- Delete data.

```
-- Delete a masking policy.
gaussdb=# DROP MASKING POLICY maskpol1, maskpol2, maskpol3, maskpol4, maskpol5, maskpol6,
maskpol7, maskpol8;

-- Delete a resource label.
gaussdb=# DROP RESOURCE LABEL mask_lb1, mask_lb2, mask_lb3, mask_lb4, mask_lb5, mask_lb6,
mask_lb7, mask_lb8;

-- Delete the tb_for_masking table.
gaussdb=# DROP TABLE tb_for_masking;

-- Delete the dev_mask and bob_mask users.
gaussdb=# DROP USER dev_mask, bob_mask;
```

Helpful Links

ALTER MASKING POLICY and DROP MASKING POLICY

7.12.8.29 CREATE MATERIALIZED VIEW

CREATE MATERIALIZED VIEW creates a complete-refresh materialized view, and you can use REFRESH MATERIALIZED VIEW (full refresh) to refresh the data in the materialized view.

CREATE MATERIALIZED VIEW is similar to CREATE TABLE AS, but it remembers the query used to initialize the view, so it can refresh data later. A materialized view has many attributes that are the same as those of a table, but does not support temporary materialized views.

Precautions

- Complete-refresh materialized views cannot be created in temporary tables or global temporary tables.

- Complete-refresh materialized views do not support NodeGroups.
- After a complete-refresh materialized view is created, most DDL operations in the base table are no longer supported.
- The IUD operation cannot be performed on complete-refresh materialized views.
- After a complete-refresh materialized view is created, if the base table data changes, you need to run the **REFRESH** command to synchronize the materialized view with the base table.
- The Ustore engine does not support the creation and use of materialized views.
- The segment-page does not support the creation and use of materialized views.

Syntax

```
CREATE MATERIALIZED VIEW mv_name
   [ (column_name [, ...] ) ]
   [ WITH ( {storage_parameter = value} [, ... ] ) ]
   [ TABLESPACE tablespace_name ]
   AS query
   [ WITH [ NO ] DATA ];
```

Parameters

- **mv_name**

  Name (optionally schema-qualified) of the materialized view to be created.

  Value range: a string. It must comply with the **naming convention**.

- **column_name**

  Specifies a column name in the new materialized view. The materialized view supports specified columns. The number of specified columns must be the same as that of columns in the result of the subsequent query statement. If no column name is provided, the column name is obtained from the output column name of the query.

  Value range: a string. It must comply with the **naming convention**.

- **WITH (storage_parameter [= value] [, ...])**

  Specifies an optional storage parameter for a table or an index. For details, see **CREATE TABLE**.

- **TABLESPACE tablespace_name**

  Tablespace to which the new materialized view belongs. If not specified, the default tablespace is used.

- **AS query**

  Specifies the **SELECT**, **TABLE**, or **VALUES** command. This query will be run in a security-constrained operation.

Examples

```
-- Create a tablespace.
gaussdb=# CREATE TABLESPACE tbs_data1 RELATIVE LOCATION 'tablespace1/tbs_data1';

-- Create an ordinary table.
gaussdb=# CREATE TABLE my_table (c1 int, c2 int)
WITH(STORAGE_TYPE=ASTORE);
```



```
-- Create a complete-refresh materialized view.
gaussdb=# CREATE MATERIALIZED VIEW my_mv AS SELECT * FROM my_table;

-- Write data to the base table.
gaussdb=# INSERT INTO my_table VALUES(1,1),(2,2);

-- Query the complete-refresh materialized view.
gaussdb=# SELECT * FROM my_mv;
 c1 | c2
----+----
(0 rows)
-- Completely refresh the complete-refresh materialized view my_mv.
gaussdb=# REFRESH MATERIALIZED VIEW my_mv;

-- Query the complete-refresh materialized view.
gaussdb=# SELECT * FROM my_mv;
 c1 | c2
----+----
  1 |  1
  2 |  2
(2 rows)
-- Delete the complete-refresh materialized view.
gaussdb=# DROP MATERIALIZED VIEW my_mv;

-- Delete the ordinary table my_table.
gaussdb=# DROP TABLE my_table;

-- Drop the tablespace.
gaussdb=# DROP TABLESPACE tbs_data1;
```

Helpful Links

ALTER MATERIALIZED VIEW, **CREATE INCREMENTAL MATERIALIZED VIEW**, **CREATE TABLE**, **DROP MATERIALIZED VIEW**, **REFRESH INCREMENTAL MATERIALIZED VIEW**, and **REFRESH MATERIALIZED VIEW**

7.12.8.30 CREATE MODEL

Description

Trains a machine learning model and saves the model.

Precautions

- The model name must be unique. Pay attention to the naming format.
- The AI training duration fluctuates greatly, and in some cases, the training duration is long. If the duration specified by the GUC parameter **statement_timeout** is too long, the training will be interrupted. You are advised to set **statement_timeout** to **0** so that the statement execution duration is not limited.

Syntax

```
CREATE  MODEL model_name USING architecture_name
FEATURES { {attribute_list} }
[TARGET attribute_name],
FROM ([schema.]table_name | subquery)
[WITH ([hyper_parameter_name = {hp_value | DEFAULT},]...)];
```


Parameters

- **model_name**

  Name of the training model, which must be unique.

  Value range: a string. It must comply with the **naming convention**.

- **architecture_name**

  Algorithm type of the training model.

  Value range: a string. Currently, the value can be **logistic_regression**, **linear_regression**, **svm_classification**, or **kmeans**.

- **attribute_list**

  Enumerated input column name of the training model.

  Value range: a string. It must comply with the naming convention of data attributes.

- **attribute_name**

  Target column name of the retraining model in a supervised learning task (simple expression processing can be performed).

  Value range: a string. It must comply with the naming convention of data attributes.

- **subquery**

  Data source.

  Value range: a string. It must comply with the SQL syntax of databases.

- **hyper_parameter_name**

  Hyperparameter name of the machine learning model.

  Value range: a string. The value range varies depending on the algorithm. For details, see "Hyperparameters supported by operators" in section "DB4AI: Database-driven AI > Native DB4AI Engine" in *Feature Guide*.

- **hp_value**

  Hyperparameter value.

  Value range: a string. The value range varies depending on the algorithm. For details, see "Default values and value ranges of hyperparameters" in section "DB4AI: Database-driven AI > Native DB4AI Engine" in *Feature Guide*.

Examples

```
-- Create a data table.
gaussdb=# CREATE TABLE houses (
id INTEGER,
tax INTEGER,
bedroom INTEGER,
bath DOUBLE PRECISION,
price INTEGER,
size INTEGER,
lot INTEGER,
mark text
);

-- Insert training data.
gaussdb=# INSERT INTO houses(id, tax, bedroom, bath, price, size, lot, mark) VALUES
(1,590,2,1,50000,770,22100,'a+'),
(2,1050,3,2,85000,1410,12000,'a+'),
(3,20,2,1,22500,1060,3500,'a-'),
(4,870,2,2,90000,1300,17500,'a+'),
```



```
(5,1320,3,2,133000,1500,30000,'a+'),
(6,1350,2,1,90500,850,25700,'a-'),
(7,2790,3,2.5,260000,2130,25000,'a+'),
(8,680,2,1,142500,1170,22000,'a-'),
(9,1840,3,2,160000,1500,19000,'a+'),
(10,3680,4,2,240000,2790,20000,'a-'),
(11,1660,3,1,87000,1030,17500,'a+'),
(12,1620,3,2,118500,1250,20000,'a-'),
(13,3100,3,2,140000,1760,38000,'a+'),
(14,2090,2,3,148000,1550,14000,'a-'),
(15,650,3,1.5,65000,1450,12000,'a-');

-- Train the model.
gaussdb=# CREATE MODEL price_model USING logistic_regression
 FEATURES size, lot
 TARGET mark
 FROM HOUSES
 WITH learning_rate=0.88, max_iterations=default;

-- Delete the model.
gaussdb=# DROP MODEL price_model;

-- Delete the table.
gaussdb=# DROP TABLE houses;
```

Helpful Links

DROP MODEL and **PREDICT BY**

7.12.8.31 CREATE OPERATOR

Description

CREATE OPERATOR defines a new operator.

Precautions

CREATE OPERATOR defines a new name operator. The user who defines the operator becomes the owner of the operator. If a schema name is given, the operator is created in the specified schema. Otherwise, it will be created in the current schema.

The operator name is a character string consisting of the following characters:

+ - * / < > = ~ ! @ # % ^ & | ` ?

When selecting a name, note the following restrictions:

- -- and /* cannot appear anywhere in the operator name, because they are regarded as the beginning of a comment.
- A multi-character operator cannot end with + or - unless the name contains at least one of the following characters:

  ~ ! @ # % ^ & | ` ?
- => The operator name is no longer used.

Operator! = is mapped to <> when being entered. Therefore, the two names are always equivalent.

At least one LEFTARG and one RIGHTARG must be defined. For binocular operators, both need to be defined. For the right operator, only LEFTARG needs to be defined. For the left operator, only RIGHTARG needs to be defined.

Also, the function_name procedure must have been defined with CREATE FUNCTION, and must be defined to accept the correct number of specified type parameters (one or two).

Other clauses declare optional operator optimization clauses. The meanings are defined in **Section 35.13**.

To create an operator, you must have the USAGE permission on the parameter type and return type, and the EXECUTE permission on the underlying function. If exchange or negative operators are specified, you must have them.

Syntax

```
CREATE OPERATOR name (
    PROCEDURE = function_name
    [, LEFTARG = left_type ] [, RIGHTARG = right_type ]
    [, COMMUTATOR = com_op ] [, NEGATOR = neg_op ]
    [, RESTRICT = res_proc ] [, JOIN = join_proc ]
    [, HASHES ] [, MERGES ]
);
```

Parameters

- **name**

  Operator to be defined. The available characters are listed above. The name can be schema-qualified, for example, CREATE OPERATOR myschema.+ (...). If there is no schema, the operator is created in the current schema. Two operators in the same schema can have the same name as long as they operate on different data types. This is a reloading process.

- **function_name**

  Function used to implement the operator.

- **left_type**

  Parameter data type on the left of the operator, if any. This parameter can be omitted if the left operator is used.

- **right_type**

  Parameter data type on the right of the operator, if any. This parameter can be omitted if the right-view operator is used.

- **com_op**

  Exchange operator corresponding to the operator.

- **neg_op**

  Negative operator corresponding to the operator.

- **res_proc**

  This operator constrains the selectivity evaluation function.

- **join_proc**

  This operator joins the selectivity evaluation function.

- **HASHES**

  Indicates that the operator supports hash joins.

- **MERGES**

  Indicates that this operator supports a merge join.

  Use the OPERATOR() syntax to provide a schema-qualified operator name in com_op or other optional parameters. For example:

  ```
  COMMUTATOR = OPERATOR(myschema.===) ,
  ```

Examples

The following command defines a new operator: equal area for the box data type.

```
CREATE OPERATOR === (
    LEFTARG = box,
    RIGHTARG = box,
    PROCEDURE = area_equal_procedure,
    COMMUTATOR = ===,
    NEGATOR = !==,
    RESTRICT = area_restriction_procedure,
    JOIN = area_join_procedure,
    HASHES, MERGES
);
SELECT box1 === box2;
```

Define operators in other schemas.

```
CREATE OPERATOR pg_temp.=== (
    LEFTARG = box,
    RIGHTARG = box,
    PROCEDURE = area_equal_procedure,
    COMMUTATOR = ===,
    NEGATOR = !==,
    RESTRICT = area_restriction_procedure,
    JOIN = area_join_procedure,
    HASHES, MERGES
);
SELECT box1 OPERATOR(pg_temp.===) box2;
```

Helpful Links

ALTER OPERATOR, **CREATE OPERATOR CLASS**, and **DROP OPERATOR**

7.12.8.32 CREATE OPERATOR CLASS

Description

Defines a new operator class. This function is for internal use only. You are advised not to use it.

Precautions

CREATE OPERATOR CLASS defines a new operator class. An operator class defines how to use a specified data type together with an index. The operator class declares that certain operators will provide particular roles or "strategies" for this data type and this index method. The operator class also declares the supported programs used by the index method when the operator class is selected for an index column. All the operators and functions used by an operator class must be defined before the operator class is created.

If a schema name is given, then the operator class is created in the specified schema. Otherwise, it is created in the current schema. Two operator classes in the

same schema can have the same name only if they are for different index methods.

The user who defines an operator class becomes the owner. Currently, the creator must be an initial user.

CREATE OPERATOR CLASS does not check whether the class definition includes all the operators and functions required by the index method, nor whether the operators and functions form a self-contained set.

Related operator classes can be grouped into operator families. To add a new operator class to an existing family, specify the **FAMILY** option in CREATE OPERATOR CLASS. Without this option, the new class is placed into a family with the same name (a family is created if it does not exist).

Syntax

```
CREATE OPERATOR CLASS
name [ DEFAULT ] FOR TYPE data_type
USING index_method [
FAMILY family_name ] AS
{  OPERATOR strategy_number operator_name [ (
op_type, op_type ) ] [ FOR SEARCH | FOR ORDER BY sort_family_name ]
| FUNCTION
support_number [ ( op_type [ , op_type ] ) ] function_name ( argument_type [,
...] )
| STORAGE storage_type
} [, ... ];
```

Parameters

- **name**

  Specifies the name of the operator class to be created (which can be schema-qualified).

- **default**

  Specifies that the operator class will become the default operator class for its data type. At most one operator class can be the default for a specific data type and index method.

- **data_type**

  Specifies the column data type processed by the operator class.

- **index_method**

  Specifies the name of the index method processed by the operator class.

- **family_name**

  Specifies the name of an existing operator family where an operator class is added. If not specified, the class is placed into a family with the same name (a family is created if it does not exist).

- **strategy_number**

  Specifies the strategy number of the index method associated with the operator class.

- **operator_name**

  Specifies the name (which can be schema-qualified) of the operator associated with the operator class.

- **op_type**

Specifies the data type of the operator's operand in an OPERATOR clause. The value **NONE** represents a left unary operator or right unary operator. The data type of the operand can be omitted if it is the same as that of the operator class.

In a FUNCTION clause, if the data type of the function's operand is different from the input data type of the function (such as B-tree comparison functions and hash functions) or the class's data type, the data type of the operand supported by this function must be included in this clause. These default values are correct. Therefore, **op_type** need not be specified in FUNCTION clauses, except in cases where the B-tree sort support function supports cross-type comparisons.

- **sort_family_name**

  Specifies the name of an existing B-tree operator family that describes the sort ordering associated with a sort operator.

  The default value is **FOR SEARCH**.

- **support_number**

  Specifies the number of an index method for a function associated with the operator class.

- **function_name**

  Specifies a function name of an index method for an operator class.

- **argument_type**

  Specifies a data type of a function parameter.

- **storage_type**

  Specifies the data type stored in the index. Normally, this is the same as the column data type, but some index methods allow it to be different. The STORAGE clause must be omitted unless the index method allows a different type to be used.

Examples

```
-- Define a function.
CREATE FUNCTION func_add_sql(num1 integer, num2 integer) RETURN integer AS BEGIN
RETURN num1 + num2;
END;
/
-- Create an operator class and associate it with the preceding function.
CREATE OPERATOR CLASS oc1 DEFAULT FOR TYPE _int4 USING btree AS FUNCTION 1 func_add_sql
(integer, integer);
```

Helpful Links

ALTER OPERATOR, **CREATE OPERATOR**, and **DROP OPERATOR**

7.12.8.33 CREATE PACKAGE

Description

Creates a package.

Precautions

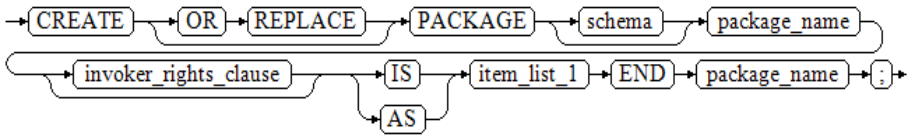
- The functions or stored procedures declared in the package specification must be defined in the package body.
- When a stored procedure is created, a write lock is added only to the CREATE stored procedure or package, and a read lock is added only to the functions and packages on which the functions depend during compilation and execution.
- During instantiation, the stored procedure with COMMIT or ROLLBACK cannot be called.
- Package functions cannot be called in triggers.
- Variables in a package cannot be directly used in external SQL statements.
- Private variables and stored procedures in a package cannot be called outside the package.
- Usage that stored procedures do not support are not supported. For example, if COMMIT or ROLLBACK cannot be called in a function, COMMIT or ROLLBACK cannot be called in the function of a package.
- The name of a schema cannot be the same as that of a package.
- Only A-version stored procedures and function definitions are supported.
- Variables with the same name in a package, including parameters with the same name in a package, are not supported.
- The global variables in a package are at the session level. The variables in packages cannot be shared in different sessions.
- When a function of an autonomous transaction is called in a package, the cursor variables in the package and recursive functions that use the cursor variables in the package are not allowed.
- The package does not declare the ref cursor variables.
- The default permission on a package is SECURITY INVOKER. To change the default permission to SECURITY DEFINER, set the GUC parameter **behavior_compat_options** to **'plsql_security_definer'**.
- A user granted with the **CREATE ANY PACKAGE** permission can create packages in the public and user schemas.
- If the name of a package to be created contains special characters, the special characters cannot contain spaces. You are advised to set the GUC parameter **behavior_compat_options** to **"skip_insert_gs_source"**. Otherwise, an error may occur.
- When a package is created, it depends on an undefined object. If **behavior_compat_options** is set to **'plpgsql_dependency'**, the creation can be executed and a warning message is displayed. If **behavior_compat_options** is not set to **'plpgsql_dependency'**, the creation cannot be executed.
- If a view directly depends on an A-compatible function in a package and the **behavior_compat_options** parameter is set to **'plpgsql_dependency'**, the view can be accessed if the package is created again. However, if the **behavior_compat_options** parameter is not set to **'plpgsql_dependency'**, the view cannot be accessed.
- When you create a package function, the default parameter value can contain variables in the package but cannot contain cross-package variables.

- When the complex function is called in a package, for example, func(x).a, a composite type is returned. Cross-schema call is supported, but the database.schema.package.func(x).b call is not supported.
- When a stored procedure in a package is created, if the stored procedure name is in the *schema.func* or *package.func* format, only the value of *func* is obtained, and the declaration of *schema* or *package* is invalid. To disable this behavior by default, set the GUC parameter **behavior_compat_options** to **'forbid_package_function_with_prefix'**.
- Cursors with parameters created in a package can be referenced (%RowType) and opened across packages. The restrictions are as follows:
  a. Currently, cursors cannot be called in database.schema.package.cursor mode.
  b. If a cursor is defined as a function parameter, you need to set **behavior_compat_options** to **'allow_procedure_compile_check'** in advance to support the parsing of the cursor type.
  c. If **behavior_compat_options** is not set to **'allow_procedure_compile_check'**, cursor%RowType is processed as a composite type, that is, a record. After the parameter is set, the specific type of the cursor can be parsed.

Syntax

- CREATE PACKAGE SPECIFICATION

```
CREATE [ OR REPLACE ] PACKAGE [ schema ] package_name
  [ invoker_rights_clause ] { IS | AS } item_list_1 END package_name;
```

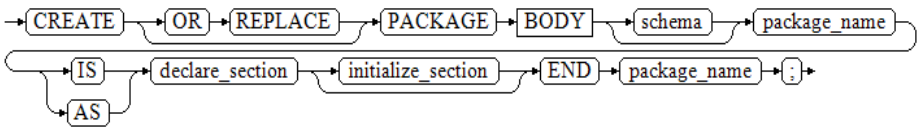


  – invoker_rights_clause can be declared as AUTHID DEFINER or AUTHID CURRENT_USER, which indicate the definer permission and caller permission, respectively.
  – **item_list_1** can be a declared variable, stored procedure, or function.

  PACKAGE SPECIFICATION (header) declares public variables, functions, and exceptions in a package, which can be called by external functions or stored procedures. It can only declare stored procedures and functions but cannot define them.

- CREATE PACKAGE BODY

```
CREATE [ OR REPLACE ] PACKAGE BODY [ schema ] package_name
  { IS | AS } declare_section [ initialize_section ] END package_name;
```



  PACKAGE BODY defines private variables and functions in a package. If a variable or function is not declared by PACKAGE SPECIFICATION, it is a private variable or private function.

  The package body also has an initialization part to initialize the package. For details, see the example.

Examples

- Create a package.

```
-- Create a test table tbl_test.
gaussdb=# CREATE TABLE tbl_test(uid varchar(6) PRIMARY KEY, area_id varchar(5), level int);

-- Create a package header.
gaussdb=# CREATE OR REPLACE PACKAGE pkg_test AS
    -- Public stored procedures can be called by external systems.
    PROCEDURE proc_ist_tbl_test(v_uid in varchar, v_area_id varchar, v_level int);
    PROCEDURE proc_del_tbl_test(v_uid in varchar);
    PROCEDURE proc_upd_tbl_test(v_uid in varchar, v_area_id varchar, v_level int);
END pkg_test;
/

-- Create a package body.
gaussdb=# CREATE OR REPLACE PACKAGE BODY pkg_test AS
    PROCEDURE proc_ist_tbl_test(v_uid in varchar, v_area_id varchar, v_level int) AS
    BEGIN
        INSERT INTO tbl_test VALUES (v_uid, v_area_id, v_level);
    END;
    PROCEDURE proc_del_tbl_test(v_uid in varchar) AS
    BEGIN
        DELETE FROM tbl_test WHERE uid = v_uid;
    END;
    PROCEDURE proc_upd_tbl_test(v_uid in varchar, v_area_id varchar, v_level int) AS
    BEGIN
        UPDATE tbl_test SET area_id = v_area_id, level = v_level WHERE uid = v_uid;
    END;
    var4 int := 10;
    -- The instantiation starts.
BEGIN
    var4 := 10;
    dbe_output.print_line(var4);
END pkg_test;
/
```

- Call a package.

```
-- Use CALL to call the stored procedure of a package.
gaussdb=# CALL pkg_test.proc_ist_tbl_test('0A00B1','01001',24);
gaussdb=# SELECT * FROM tbl_test;
  uid   | area_id | level
--------+---------+---------
 0A00B1 | 01001   |     24
(1 row)

-- Use SELECT to call the stored procedure of a package.
gaussdb=# SELECT pkg_test.proc_upd_tbl_test('0A00B1','01001','26');
gaussdb=# SELECT * FROM tbl_test;
  uid   | area_id | level
--------+---------+-------
 0A00B1 | 01001   |    26
(1 row)

-- Call a stored procedure of a package in an anonymous block.
gaussdb=# BEGIN
pkg_test.proc_del_tbl_test('0A00B1');
END;
/
gaussdb=# SELECT * FROM tbl_test;
 uid | area_id | level
-----+---------+-------
(0 rows)

-- Delete.
gaussdb=# DROP TABLE tbl_test;
gaussdb=# DROP PACKAGE pkg_test;
```


Helpful Links

ALTER PACKAGE and **DROP PACKAGE**

7.12.8.34 CREATE PROCEDURE

Description

Creates a stored procedure.

Precautions

- If the parameters or return values of a stored procedure have precision, the precision is not checked.
- When creating a stored procedure, you are advised to explicitly specify the schemas of all operations on table objects in the stored procedure definition. Otherwise, the stored procedure may fail to be executed.
- When a stored procedure is created, a write lock is added only to the CREATE stored procedure or package, and a read lock is added only to the functions and packages on which the functions depend during compilation and execution.
- **current_schema** and **search_path** specified by SET during stored procedure creation are invalid. **search_path** and **current_schema** before and after function execution should be the same.
- When the function is called by SELECT or CALL, an argument must be provided in the output parameter. The argument does not take effect.
- A stored procedure with the **PACKAGE** attribute can use overloaded functions.
- Do not create an overloaded stored procedure with different parameter names but same stored procedure name and parameter list type.
- When an overloaded stored procedure is called, the variable type must be specified.
- Do not create a stored procedure that has the same name and parameter list as the function.
- Do not overload stored procedures with different default values.
- Only the in, out, and inout parameters of the stored procedure cannot be reloaded after the GUC parameter **behavior_compat_options='proc_outparam_override'** is enabled. They can be reloaded after the parameter is disabled.
- A-compatible functions are created for A-compatible databases and PG-compatible functions are created for PG-compatible databases. Hybrid creation is not recommended.
- If a function supports overloading, you need to add the keyword PACKAGE.
- If an undeclared variable is used in a stored procedure, an error is reported when the stored procedure is called.
- When you create a procedure, you cannot insert aggregate functions or other functions out of the average function.
- The stored procedure does not support operations that will return a set.

- When stored procedures without parameters are called in another stored procedure, you can omit brackets and call stored procedures using their names directly.
- When other functions with output parameters are called in a stored procedure and an assignment expression, set the GUC parameter **set behavior_compat_options** to **'proc_outparam_override'**, define variables of the same type as the output parameters in advance, and use the variables as output parameters to call other functions with output parameters for the output parameters to take effect. Otherwise, the output parameters of the called function will be ignored.
- If the **out** parameter is used as the output parameter in an expression, the expression does not take effect in the following scenarios: The execute immediate sqlv using func syntax is used to execute a function. The select func into syntax is used to execute a function. DML statements such as INSERT and UPDATE are used to execute a function. When a function with the **out** output parameter is used as an input parameter, that is, **fun (func (out b), a)**, the **out b** parameter does not take effect.
- The stored procedure supports viewing, exporting, and importing parameter comments.
- The stored procedure supports viewing, exporting, and importing parameter comments between IS/AS and plsql_body.
- The default permission on a stored procedure is SECURITY INVOKER. To change the default permission to SECURITY DEFINER, set the GUC parameter **behavior_compat_options** to **'plsql_security_definer'**.
- Users granted with the CREATE ANY FUNCTION permission can create or replace stored procedures in the user schemas.
- **out/inout** must be set to a variable but not a constant.
- In a centralized environment, if you want to call a stored procedure with the same in parameters but different out parameters, you need to set the GUC parameter **behavior_compat_options** to **'proc_outparam_override'**. After the parameter is enabled, you must add the out parameters no matter whether you use the SELECT or CALL statement to call the stored procedure. After the parameter is enabled, you cannot use **perform** to call a stored procedure or function.
- When a stored procedure is created, it depends on an undefined object. If **behavior_compat_options** is set to **'plpgsql_dependency'**, the creation can be executed, and a warning message is displayed. If **behavior_compat_options** is not set to **'plpgsql_dependency'**, the creation cannot be executed.
- When separation of duties is enabled, the stored procedure of the definer permission can be re-created only by the current user.
- If a stored procedure with the definer specified is created in a schema of another user, the stored procedure will be executed by another user, which may cause unauthorized operations. Therefore, exercise caution when performing this operation.
- When the complex function is called in a stored procedure, for example, **func(x).a**, a composite type is returned. Cross-schema call is supported, but the **database.schema.package.func(x).b** call is not supported.
- When a stored procedure with the out parameter is called, you can set the GUC parameter **set behavior_compat_options** to

'proc_outparam_transfer_length' to transfer the parameter length. The specifications and constraints are as follows:

a. The following types are supported: CHAR(n), CHARACTER(n), NCHAR(n), VARCHAR(n), VARYING(n), VARCHAR2(n), and NVARCHAR2(n).

b. If the out parameter does not take effect (for example, **perform**), the length does not need to be transferred.

c. The following types do not support precision transfer: NUMERIC, DECIMAL, NUMBER, FLOAT, DEC, INTEGER, TIME, TIMESTAMP, INTERVAL, TIME WITH TIME ZONE, TIMESTAMP WITH TIME ZONE, TIME WITHOUT TIME ZONE, and TIMESTAMP WITHOUT TIME ZONE.

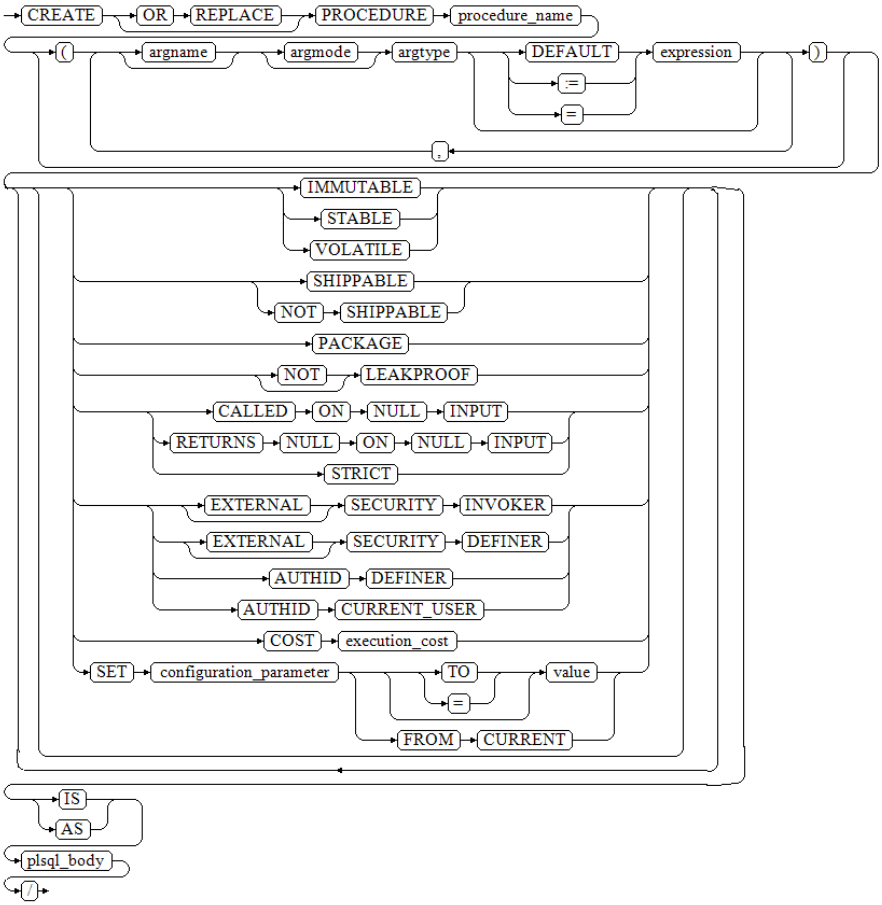
d. The parameter length can be transferred regardless of whether the GUC parameter **set behavior_compat_options** is set to **proc_outparam_override**.

e. To transfer the length of elements of the collection type and the length of elements of the array type nested by the collection type, you need to set the GUC parameter **behavior_compat_options** to **tableof_elem_constraints**.

- Functions contain syntaxes and functions that use GUC parameters to control features. If GUC parameters are modified in a session, the function may retain the behavior before the modification. Therefore, exercise caution when modifying GUC parameters.

Syntax

```
CREATE [ OR REPLACE ] PROCEDURE procedure_name
   [ ( {[ argname ] [ argmode ] argtype [ { DEFAULT | := | = } expression ]}[,...]) ]
   [
     { IMMUTABLE | STABLE | VOLATILE }
     | { SHIPPABLE | NOT SHIPPABLE }
     | {PACKAGE}
     | [ NOT ] LEAKPROOF
     | { CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
     | {[ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER | AUTHID DEFINER | AUTHID
CURRENT_USER}
     | COST execution_cost
     | SET configuration_parameter { [ TO | = ] value | FROM CURRENT }
   ][ ... ]
 { IS | AS }
plsql_body
/
```



Parameters

- **OR REPLACE**

  Replaces the original definition when two stored procedures are with the same name.

- **procedure_name**

  Specifies the name of the stored procedure that is created (optionally with schema names).

  Value range: a string. It must comply with the **naming convention**.

  NOTE

  When creating a function with the same name as a system function, you need to specify the schema of the function.

- **argmode**

  Specifies the mode of an argument.

NOTICE

VARIADIC specifies parameters of the array type.

Value range: **IN**, **OUT**, **INOUT**, and **VARIADIC**. The default value is **IN**. Only the parameters in **OUT** mode can follow the **VARIADIC** parameter.

- **argname**

  Specifies the argument name.

  Value range: a string. It must comply with the **naming convention**.

- **argtype**

  Specifies the type of an argument. **%TYPE** or **%ROWTYPE** can be used to indirectly reference a variable or table type. For details, see **Variable Definition Statements**.

  Value range: a valid data type

  NOTE

  In **PROCEDURE argtype** outside **PACKAGE**, **%TYPE** cannot reference the type of the *PACKAGE* variable.

- **expression**

  Specifies the default expression of a parameter.

  NOTE

  - If **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s2**, the default expression is not supported when the parameter is in INOUT mode.
  - It is recommended that you define all default parameters after all non-default parameters.
  - If a function with default parameters is called, input parameters are added to the function from left to right. If inputs of non-default parameters are missing, an error is reported.
  - If **proc_uncheck_default_param** is enabled and a function with default parameters is called, input parameters are added to the function from left to right. The number of defaulted inputs depends on the number of default parameters. If an input of a non-default parameter is missing, the previous default value is used to fill this parameter.
  - When **a_format_version** is set to **10c**, **a_format_dev_version** is set to **s1**, **proc_outparam_override** is disabled, and the function parameters include the output parameter **out** and **default**, the default value cannot be used.

- **configuration_parameter**
  - **value**

    Sets the specified configuration parameter to a specified value. If the **value** is **DEFAULT**, the default setting is used in the new session. **OFF** disables the setting.

    Value range: a string

    - DEFAULT
    - OFF
    - Specified default value
  - **from current**

    Uses the value of **configuration_parameter** of the current session.

- **IMMUTABLE, STABLE,**...

  Specifies a constraint. The function of each parameter is similar to that of CREATE FUNCTION. For details, see **CREATE FUNCTION**.

- **plsql_body**

  Specifies the PL/SQL stored procedure body.

  NOTICE

  When you perform password-related operations, such as user creation, password change, and encryption/decryption, in a stored procedure, the password will be recorded in the system catalogs and logs in plaintext. To prevent sensitive information leakage, you are advised not to perform password-related operations in a stored procedure.

  NOTE

  No specific order is applied to **argname** and **argname**. The following order is advised: **argname**, **argmode**, and **argtype**.

Examples

- Create a stored procedure.

```
-- Create a stored procedure and print the sum of the input parameters.
gaussdb=# CREATE OR REPLACE PROCEDURE proc_add(i int, j int)
AS
BEGIN
    dbe_output.print_line('result is: '|| i+j);
END;
/

-- Use CALL to call a stored procedure.
gaussdb=# CALL proc_add(16,17);

-- Use a program block to call a stored procedure.
gaussdb=# BEGIN
   proc_add(16,17);
END;
/

-- Delete.
gaussdb=# DROP PROCEDURE proc_add;
```

- Create a stored procedure whose parameter model is **VARIADIC**.

```
-- Create the stored procedure pro_variadic.
gaussdb=# CREATE OR REPLACE PROCEDURE pro_variadic (var1 VARCHAR2(10) DEFAULT 'hello!',var4
VARIADIC int4[])
AS
BEGIN
   dbe_output.print_line(var1);
END;
/

-- Execute the stored procedure.
gaussdb=# SELECT pro_variadic(var1=>'hello', VARIADIC var4=> array[1,2,3,4]);

-- Delete.
gaussdb=# DROP PROCEDURE pro_variadic;
```

- Parameter models: **IN** and **OUT**
  - **IN** (default) indicates that the parameter is an input parameter.
  - **OUT** indicates that the parameter is an output parameter.
  - **IN OUT** indicates that the parameter is an input and output parameter

```
-- Create the stored procedure proc_add1. num1 and num2 are input parameters, and num3 is an
output parameter.
```



```
gaussdb=# CREATE PROCEDURE proc_add1 (num1 in int, num2 in int, num3 out int)
AS
BEGIN
    num3 := num1 + num2;
END;
/

-- Use a program block to call the stored procedure and use variable c to receive the parameters sent by the stored procedure.
gaussdb=# DECLARE
    a int := 20;
    b int := 32;
    c int := 0;
BEGIN
    proc_add1(a,b,c);
    dbe_output.put_line(c);
END;
/

-- Delete.
gaussdb=# DROP PROCEDURE proc_add1;
```

Helpful Links

ALTER PROCEDURE and **DROP PROCEDURE**

Suggestions

- analyse | analyze
  - Do not run **ANALYZE** in a transaction or anonymous block.
  - Do not run **ANALYZE** in a function or stored procedure.

7.12.8.35 CREATE RESOURCE LABEL

Description

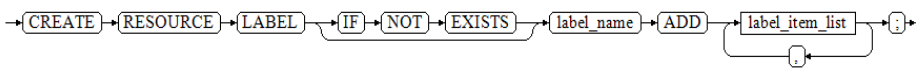
CREATE RESOURCE LABEL is used to create a resource label.

Precautions

Only users with the POLADMIN or SYSADMIN permission, or the initial user can perform this operation.

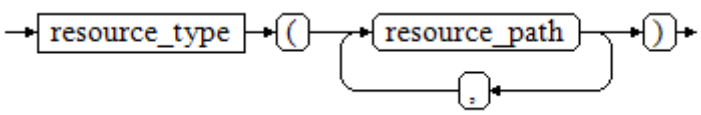
Syntax

```
CREATE RESOURCE LABEL [IF NOT EXISTS] label_name ADD label_item_list[, ...];
```



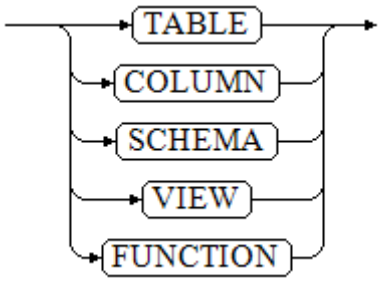
- label_item_list

  ```
  resource_type(resource_path[, ...])
  ```



- resource_type

  ```
  { TABLE | COLUMN | SCHEMA | VIEW | FUNCTION }
  ```

Parameters

- **IF NOT EXISTS**

  If a resource label with the same name already exists, no error is generated. Instead, a message is displayed, indicating that the resource label already exists.

- **label_name**

  Specifies the resource label name, which must be unique.

  Value range: a string. It must comply with the **naming convention**.

- **resource_type**

  Specifies the type of database resources to be labeled.

  Value range: **TABLE**, **COLUMN**, **SCHEMA**, **VIEW**, and **FUNCTION**.

- **resource_path**

  Specifies the path of database resources.

Examples

```
-- Create table tb_for_label.
gaussdb=# CREATE TABLE tb_for_label(col1 text, col2 text, col3 text);

-- Create a resource label based on a table.
gaussdb=# CREATE RESOURCE LABEL IF NOT EXISTS table_label add TABLE(public.tb_for_label);

-- Create an existing table resource label again and compare the differences between adding IF NOT EXISTS and not adding IF NOT EXISTS.
gaussdb=# CREATE RESOURCE LABEL IF NOT EXISTS table_label add TABLE(public.tb_for_label);
NOTICE:  table_label label already defined, skipping
CREATE RESOURCE LABEL
gaussdb=# CREATE RESOURCE LABEL table_label add TABLE(public.tb_for_label);
ERROR:  table_label label already defined

-- Create a resource label based on columns.
gaussdb=# CREATE RESOURCE LABEL IF NOT EXISTS column_label add COLUMN(public.tb_for_label.col1);

-- Create schema schema_for_label.
gaussdb=# CREATE SCHEMA schema_for_label;

-- Create a resource label based on a schema.
gaussdb=# CREATE RESOURCE LABEL IF NOT EXISTS schema_label add SCHEMA(schema_for_label);

-- Create view view_for_label.
gaussdb=# CREATE VIEW view_for_label AS SELECT 1;

-- Create a resource label based on a view.
gaussdb=# CREATE RESOURCE LABEL IF NOT EXISTS view_label add VIEW(view_for_label);

-- Create function func_for_label.
gaussdb=# CREATE FUNCTION func_for_label RETURNS TEXT AS $$ SELECT col1 FROM tb_for_label; $$
```



```
LANGUAGE SQL;

-- Create a resource label based on a function.
gaussdb=# CREATE RESOURCE LABEL IF NOT EXISTS func_label add FUNCTION(func_for_label);

-- Delete the table resource label table_label.
gaussdb=# DROP RESOURCE LABEL IF EXISTS table_label;

-- Delete the column resource label column_label.
gaussdb=# DROP RESOURCE LABEL IF EXISTS column_label;

-- Delete the function resource label func_for_label.
gaussdb=# DROP FUNCTION func_for_label;

-- Delete the view resource label view_for_label.
gaussdb=# DROP VIEW view_for_label;

-- Delete the schema resource label schema_for_label.
gaussdb=# DROP SCHEMA schema_for_label;

-- Delete the tb_for_label table.
gaussdb=# DROP TABLE tb_for_label;
```

Helpful Links

ALTER RESOURCE LABEL and DROP RESOURCE LABEL

7.12.8.36 CREATE RESOURCE POOL

Description

CREATE RESOURCE POOL creates a resource pool and specifies the Cgroup of the resource pool.

Precautions

Only users with the SYSADMIN permission, or the initial user can perform this operation.

Syntax

```
CREATE RESOURCE POOL pool_name
    [WITH ({MEM_PERCENT=pct | CONTROL_GROUP="group_name" | ACTIVE_STATEMENTS=stmt |
MAX_DOP = dop | MEMORY_LIMIT='memory_size' | io_limits=io_limits | io_priority='priority' |
nodegroup='nodegroup_name' | is_foreign = boolean }[, ... ])];
```

Parameters

- **pool_name**

  Specifies the name of a resource pool.

  The name of the resource pool cannot be same as that of an existing resource pool.

  Value range: a string. It must comply with the **naming convention**.

- **group_name**

  Specifies the name of a Cgroup.

NOTE

- You can use either double quotation marks ("") or single quotation marks (') in the syntax when setting the name of a Cgroup.
- The value of **group_name** is case-sensitive.
- If an administrator specifies a Workload Cgroup under Class, for example, **control_group** set to **class1:workload1**, the resource pool will be associated with the **workload1** Cgroup under **class1**. The level of **Workload** can also be specified. For example, **control_group** is set to **class1:workload1:1**.
- If a database user specifies the Timeshare string (**Rush**, **High**, **Medium**, or **Low**) in the syntax, for example, **control_group** is set to **High**, the resource pool will be associated with the **High** Timeshare Cgroup under **DefaultClass**.

Value range: a string. It must comply with the rule in the description, which specifies the created Cgroup.

- **stmt**

  Specifies the maximum number of statements that can be concurrently executed in a resource pool.

  Value range: numeric data ranging from –1 to 2147483647 **–1** indicates no restriction and **0** indicates that no statement can be executed.

- **dop**

  Specifies the maximum statement concurrency degree for a resource pool, equivalent to the number of threads that can be created for executing a statement.

  Value range: numeric data ranging from 1 to 2147483647

- **memory_size**

  Specifies the maximum memory size of a resource pool.

  Value range: a string from 1 KB to 2047 GB

- **mem_percent**

  Specifies the proportion of available resource pool memory to the total memory or group user memory.

  In multi-tenant scenarios, **mem_percent** of group users or service users ranges from 1 to 100. The default value is **20**.

  In common scenarios, **mem_percent** of common users ranges from 0 to 100. The default value is **0**.

  NOTE

  When both of **mem_percent** and **memory_limit** are specified, only **mem_percent** takes effect.

- **io_limits**

  Specifies the upper limit of IOPS in a resource pool.

  It is counted by 10 thousands per second.

- **io_priority**

  Specifies the I/O priority for jobs that consume many I/O resources. It takes effect when the I/O usage reaches 90%.

  There are three priorities: **Low**, **Medium**, and **High**. If you do not want to control I/O resources, use the default value **None**.

> NOTE
>
> The settings of **io_limits** and **io_priority** are valid only for complex jobs, such as batch import (using **INSERT INTO SELECT**, **COPY FROM**, or **CREATE TABLE AS**), complex queries involving over 500 MB data on each DN, and **VACUUM FULL**.

- **max_workers**

  Concurrency in a table during data redistribution. This column is used only for scaling.

- **max_connections**

  Maximum number of connections that can be used by a resource pool.

  > NOTE
  >
  > The total maximum number of connections in all resource pools cannot exceed the maximum number of connections specified by **max_connections** of the entire GaussDB process.

- **max_dynamic_memory**

  Specifies the maximum dynamic memory that can be used by a resource pool.

- **max_shared_memory**

  Specifies the maximum shared memory that can be used by a resource pool.

- **max_concurrency**

  Specifies the maximum concurrent requests that can be used by a resource pool.

Examples

This example assumes that you have created a Cgroup in advance. (Contact the administrator to create a Cgroup.)

```
-- Create a default resource pool, and associate it with the Medium Timeshare Cgroup under Workload under DefaultClass.
gaussdb=# CREATE RESOURCE POOL pool1;

-- Create a resource pool and specify the High Timeshare Workload Cgroup under the DefaultClass Cgroup.
gaussdb=# CREATE RESOURCE POOL pool2 WITH (CONTROL_GROUP="High");

-- Create a resource pool, and associate it with the Low Timeshare Cgroup under Workload under class1.
gaussdb=# CREATE RESOURCE POOL pool3 WITH (CONTROL_GROUP="class1:Low");

-- Create a resource pool, and associate it with the wg1 Workload Cgroup under class1.
gaussdb=# CREATE RESOURCE POOL pool4 WITH (CONTROL_GROUP="class1:wg1");

-- Create a resource pool, and associate it with the wg2 Workload Cgroup under class1.
gaussdb=# CREATE RESOURCE POOL pool5 WITH (CONTROL_GROUP="class1:wg2:3");

-- Delete the resource pool.
gaussdb=# DROP RESOURCE POOL pool1;
gaussdb=# DROP RESOURCE POOL pool2;
gaussdb=# DROP RESOURCE POOL pool3;
gaussdb=# DROP RESOURCE POOL pool4;
gaussdb=# DROP RESOURCE POOL pool5;
```

Helpful Links

ALTER RESOURCE POOL and **DROP RESOURCE POOL**

7.12.8.37 CREATE ROLE

Description

Creates a role.

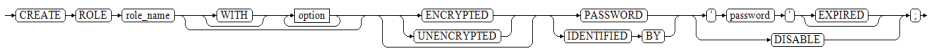
A role is an entity that owns database objects and permissions. In different environments, a role can be considered a user, a group, or both.

Precautions

- If a role is added to the database, the role does not have the login permission.
- Only the user who has the CREATE ROLE permission or a system administrator is allowed to create roles.

Syntax

```
CREATE ROLE role_name [ [ WITH ] option [ ... ] ] [ ENCRYPTED | UNENCRYPTED ] { PASSWORD |
IDENTIFIED BY } { 'password' [EXPIRED] | DISABLE };
```



The syntax of role information configuration clause **option** is as follows:

```
{SYSADMIN | NOSYSADMIN}
    | {MONADMIN | NOMONADMIN}
    | {OPRADMIN | NOOPRADMIN}
    | {POLADMIN | NOPOLADMIN}
    | {AUDITADMIN | NOAUDITADMIN}
    | {CREATEDB | NOCREATEDB}
    | {USEFT | NOUSEFT}
    | {CREATEROLE | NOCREATEROLE}
    | {INHERIT | NOINHERIT}
    | {LOGIN | NOLOGIN}
    | {REPLICATION | NOREPLICATION}
    | {PERSISTENCE | NOPERSISTENCE}
    | CONNECTION LIMIT connlimit
    | VALID BEGIN 'timestamp'
    | VALID UNTIL 'timestamp'
    | RESOURCE POOL 'respool'
    | USER GROUP 'groupuser'
    | PERM SPACE 'spacelimit'
    | TEMP SPACE 'tmpspacelimit'
    | SPILL SPACE 'spillspacelimit'
    | NODE GROUP logic_cluster_name
    | IN ROLE role_name [, ...]
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN role_name [, ...]
    | USER role_name [, ...]
    | SYSID uid
    | DEFAULT TABLESPACE tablespace_name
    | PROFILE DEFAULT
    | PROFILE profile_name
    | PGUSER
```

Parameters

- **role_name**

  Specifies the name of a role.

  Value range: a string. It must comply with the **naming convention**. A value can contain a maximum of 63 characters. If the value contains more than 63

characters, the database truncates it and retains the first 63 characters as the role name. If a role name contains uppercase letters, the database automatically converts the uppercase letters into lowercase letters. To create a role name that contains uppercase letters, enclose the role name with double quotation marks ("").

NOTE

The identifier must be letters, underscores (_), digits (0–9), or dollar signs ($) and must start with a letter (a–z) or underscore (_).

- **password**

  Specifies the login password.

  A new password must:

  – Contain at least eight characters. This is the default length.

  – Differ from the username or the username spelled backward.

  – Contain at least three of the following character types: uppercase characters, lowercase characters, digits, and special characters (limited to ~!@#$%^&*()-_=+\|[{}];:,<.>/?).

  – The password can also be a ciphertext character string that meets the format requirements. This mode is mainly used to import user data. You are advised not to use it directly. If a ciphertext password is used, the user must know the plaintext corresponding to the ciphertext password and ensure that the password meets the complexity requirements. The database does not verify the complexity of the ciphertext password. Instead, the security of the ciphertext password is ensured by the user.

  – Be enclosed by single quotation marks when a role is created.

  Value range: a character string that cannot be empty.

- **EXPIRED**

  When creating a user, you can specify the **EXPIRED** parameter to create a user whose password is invalid. The user cannot perform simple or extended queries. The statement can be executed only after the password is changed.

- **DISABLE**

  By default, you can change your password unless it is disabled. To disable the password of a user, use this parameter. After the password of a user is disabled, the password will be deleted from the system. The user can connect to the database only through external authentication, for example, Kerberos authentication. Only administrators can enable or disable a password. Common users cannot disable the password of an initial user. To enable a password, run **ALTER USER** and specify the password.

- **ENCRYPTED | UNENCRYPTED**

  Determines whether a password stored in a system catalog is encrypted. According to product security requirement, the password must be stored encrypted. Therefore, **UNENCRYPTED** is forbidden in GaussDB. If the password string has already been encrypted in the SHA256 format, it is stored as it was, regardless of whether **ENCRYPTED** or **UNENCRYPTED** is specified (since the system cannot decrypt the specified encrypted password string). This allows reloading of encrypted passwords during dump/restore.

- **SYSADMIN | NOSYSADMIN**

Specifies whether a new role is a system administrator. Roles with the SYSADMIN attribute have the highest permission.

Value range: If not specified, **NOSYSADMIN** is the default.

When separation of duties is disabled, users with the SYSADMIN permission can create users with the SYSADMIN, REPLICATION, CREATEROLE, AUDITADMIN, MONADMIN, POLADMIN, or CREATEDB permission and common users.

When separation of duties is enabled, users with the SYSADMIN permission do not have the permission to create users.

- **MONADMIN | NOMONADMIN**

  Specifies whether a role is a monitor administrator.

  Value range: If not specified, **NOMONADMIN** is the default.

- **OPRADMIN | NOOPRADMIN**

  Specifies whether a role is an O&M administrator.

  Value range: If not specified, **NOOPRADMIN** is the default.

- **POLADMIN | NOPOLADMIN**

  Specifies whether a role is a security policy administrator.

  Value range: If not specified, **NOPOLADMIN** is the default.

- **AUDITADMIN | NOAUDITADMIN**

  Specifies whether a role has the audit and management attributes.

  If not specified, **NOAUDITADMIN** is the default.

- **CREATEDB | NOCREATEDB**

  Specifies a role's permission to create databases.

  A new role does not have the permission to create databases.

  Value range: If not specified, **NOCREATEDB** is the default.

- **USEFT | NOUSEFT**

  This parameter is reserved and not used in this version.

- **CREATEROLE | NOCREATEROLE**

  Specifies whether a role will be permitted to create new roles (that is, execute **CREATE ROLE** and **CREATE USER**). A role with the CREATEROLE permission can also modify and delete other roles.

  Value range: If not specified, **NOCREATEROLE** is the default.

  When separation of duties is disabled, users with the CREATEROLE permission can create users with the CREATEROLE, AUDITADMIN, MONADMIN, POLADMIN, or CREATEDB permission and common users.

  When separation of duties is enabled, users with the CREATEROLE permission can create users with the CREATEROLE, MONADMIN, POLADMIN, or CREATEDB permission and common users.

- **INHERIT | NOINHERIT**

  Specifies whether a role "inherits" the permissions of roles in the same group. It is not recommended.

- **LOGIN | NOLOGIN**

  Specifies whether a role is allowed to log in to a database. A role having the **LOGIN** attribute can be considered as a user.

Value range: If not specified, **NOLOGIN** is the default.

- **REPLICATION | NOREPLICATION**

  Specifies whether a role is allowed to initiate streaming replication or put the system in and out of backup mode. A role having the **REPLICATION** attribute is specific to replication.

  If not specified, **NOREPLICATION** is the default.

- **PERSISTENCE | NOPERSISTENCE**

  Defines a permanent user. Only the initial user is allowed to create, modify, and delete permanent users with the PERSISTENCE attribute.

- **CONNECTION LIMIT connlimit**

  Specifies how many concurrent connections the role can make.

  NOTICE

  - The system administrator is not restricted by this parameter.
  - **connlimit** is calculated separately for each primary database node. Number of connections of the database = Value of **connlimit** x Number of normal primary database nodes.

  Value range: an integer in the range [–1, 2^{31} – 1]. The default value **–1** means no limit.

- **VALID BEGIN 'timestamp'**

  Sets the timestamp when a role takes effect. If this clause is omitted, the role has no valid start time. **timestamp** indicates the start time. The format is 'YYYY-MM-DD HH:mm:ss'.

- **VALID UNTIL 'timestamp'**

  Sets a date and time after which the role's password is no longer valid. If this clause is omitted, the role has no valid end time. **timestamp** indicates the end time. The format is 'YYYY-MM-DD HH:mm:ss'.

- **RESOURCE POOL 'respool'**

  Sets the name of resource pool used by the role. The name belongs to the system catalog **pg_resource_pool**.

- **USER GROUP 'groupuser'**

  Creates a sub-user. This function is not supported in the current version.

- **PERM SPACE 'spacelimit'**

  Sets the space available for a user.

- **TEMP SPACE 'tmpspacelimit'**

  Sets the space allocated to the temporary table of a user.

- **SPILL SPACE 'spillspacelimit'**

  Sets the operator disk flushing space of a user.

- **IN ROLE role_name**

  Lists one or more existing roles whose permissions will be inherited by a new role. It is not recommended.

- **IN GROUP role_name**

  Specifies an obsolete spelling of IN ROLE. It is not recommended.
- **ROLE role_name**

  Lists one or more existing roles which are automatically added as members of the new role.
- **ADMIN role_name**

  Similar to ROLE. However, ADMIN grants permissions of new roles to other roles.
- **USER role_name**

  Specifies an obsolete spelling of the ROLE clause.
- **SYSID uid**

  The SYSID clause is ignored.
- **DEFAULT TABLESPACE tablespace_name**

  The DEFAULT TABLESPACE clause is ignored.
- **PROFILE profile_name**

  The PROFILE clause is ignored.
- **PGUSER**

  In the current version, this attribute is reserved only for forward compatibility.

Examples

- Differences between CREATE ROLE and CREATE USER:

```
-- Run CREATE ROLE to create role test_role.
gaussdb=# CREATE ROLE test_role PASSWORD '********';
-- Run CREATE USER to create user test_user.
gaussdb=# CREATE USER test_user PASSWORD '********';

-- View the information. Roles created by CREATE ROLE are not allowed to log in to the database by
default.
gaussdb=# \du test*
            List of roles
 Role name |  Attributes  | Member of
-----------+--------------+-----------
 test_role | Cannot login | {}
 test_user |              | {}

-- Enable the test_role role to log in to the database.
gaussdb=# ALTER ROLE test_role WITH LOGIN;

gaussdb=# \du test*
           List of roles
 Role name | Attributes | Member of
-----------+------------+-----------
 test_role |            | {}
 test_user |            | {}

-- View schema information. When CREATE USER is executed to create a user, a schema with the
same name is automatically created.
gaussdb=# \dn test*
    List of schemas
   Name    |   Owner
-----------+-----------
 test_user | test_user
(1 row)

-- Delete.
gaussdb=# DROP ROLE test_role;
gaussdb=# DROP USER test_user;
```


- Create a role whose password is invalid.

```
-- Create the role test_role2 whose password is valid.
gaussdb=# CREATE ROLE test_role2 PASSWORD '********' EXPIRED;
gaussdb=# ALTER ROLE test_role2 WITH LOGIN;

-- test_role2 cannot perform any operation after logging in to the database. You can perform operations only after changing the password as prompted.
gaussdb=# SET ROLE test_role2 PASSWORD '********';
gaussdb=> \d
ERROR:  Please use "ALTER ROLE user_name IDENTIFIED BY 'password' REPLACE 'old password';" to modify the expired password of user test_role2 before operation!

-- Change the password of test_role2.
gaussdb=> ALTER ROLE test_role2 IDENTIFIED BY '********' REPLACE '********';

-- Delete.
gaussdb=> RESET ROLE;
gaussdb=# DROP ROLE test_role2;
```

- Create a role and specify the effective date and expiration date.

```
-- Create a role with its validity from January 1, 2015 to January 1, 2026.
gaussdb=# CREATE ROLE test_role3 WITH LOGIN PASSWORD '********' VALID BEGIN '2015-01-01' VALID UNTIL '2026-01-01';
-- Delete.
gaussdb=# DROP ROLE test_role3;
```

Helpful Links

SET ROLE, **ALTER ROLE**, **DROP ROLE**, and **GRANT**

7.12.8.38 CREATE ROW LEVEL SECURITY POLICY

Description

Creates a row-level security policy for a table.

The policy takes effect only after row-level security is enabled (by running **ALTER TABLE ... ENABLE ROW LEVEL SECURITY**). Otherwise, this statement does not take effect.

Currently, row-level security affects the read (SELECT, UPDATE, and DELETE) of data tables and does not affect the write (INSERT and MERGE INTO) of data tables. The table owner or system administrators can create an expression in the USING clause. When the client reads the data table, the database server combines the expressions that meet the condition and applies it to the execution plan in the statement rewriting phase of a query. For each tuple in a data table, if the USING expression returns **TRUE**, the tuple is visible to the current user; if the USING expression returns **FALSE** or **NULL**, the tuple is invisible to the current user.

A row-level security policy name is specific to a table. A data table cannot have row-level security policies with the same name. Different data tables can have the same row-level security policy.

Row-level security policies can be applied to specified operations (**SELECT**, **UPDATE**, **DELETE**, and **ALL**). **ALL** indicates that SELECT, UPDATE, and DELETE will be affected. For a new row-level security policy, the default value **ALL** will be used if you do not specify the operations that will be affected.

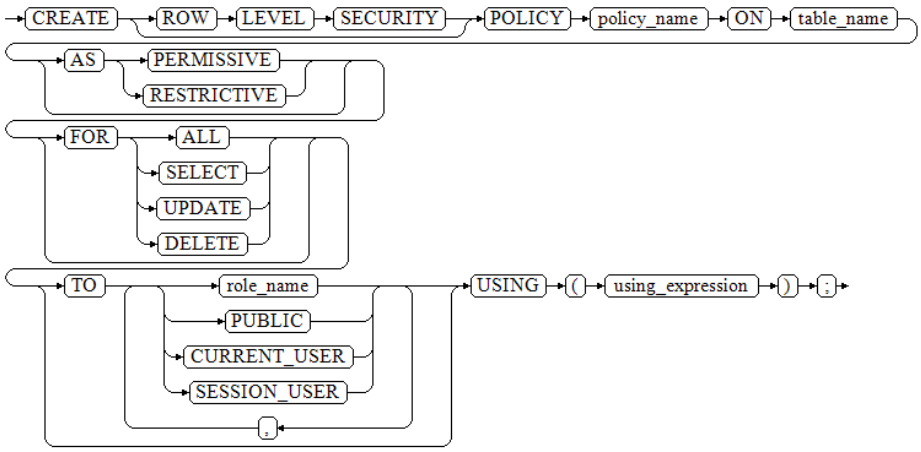
Row-level security policies can be applied to a specified user (role) or to all users (**PUBLIC**). For a new row-level security policy, the default value **PUBLIC** will be used if you do not specify the user who will be affected.

Precautions

- Row-level security policies can be defined for row-store tables, row-store partitioned tables, unlogged tables, and hash tables.
- Row-level security policies cannot be defined for foreign tables and local temporary tables.
- Row-level security policies cannot be defined for views.
- A maximum of 100 row-level security policies can be defined for a table.
- System administrators are not affected by row-level security policies and can view all data in a table.
- Tables queried by using SQL statements, views, functions, and stored procedures are affected by row-level security policies.
- The data type of a table column to which a row-level security policy has been added cannot be changed.

Syntax

```
CREATE [ ROW LEVEL SECURITY ] POLICY policy_name ON table_name
    [ AS { PERMISSIVE | RESTRICTIVE } ]
    [ FOR { ALL | SELECT | UPDATE | DELETE } ]
    [ TO { role_name | PUBLIC | CURRENT_USER | SESSION_USER } [, ...] ]
    USING ( using_expression );
```



Parameters

- **policy_name**

  Specifies the name of a row-level security policy to be created. The names of row-level security policies for a table must be unique.

- **table_name**

  Specifies the name of a table to which a row-level security policy is applied.

- **PERMISSIVE | RESTRICTIVE**

  PERMISSIVE enables the permissive policy for row-level security. The conditions of the permissive policy are joined through the OR expression.

  RESTRICTIVE enables the restrictive policy for row-level security. The conditions of the restrictive policy are joined through the AND expression. The join methods are as follows:

  ```
  (using_expression_permissive_1 OR using_expression_permissive_2 ...) AND
  (using_expression_restrictive_1 AND using_expression_restrictive_2 ...)
  ```


The default value is **PERMISSIVE**.

- **command**

  Specifies the SQL operations affected by a row-level security policy, including **ALL**, **SELECT**, **UPDATE**, and **DELETE**. If this parameter is not specified, the default value **ALL** will be used, covering SELECT, UPDATE, and DELETE.

  If **command** is set to **SELECT**, only tuple data that meets the condition (the return value of **using_expression** is **TRUE**) can be queried. The operations that are affected include SELECT, SELECT FOR UPDATE/SHARE, UPDATE ... RETURNING, and DELETE ... RETURNING.

  If **command** is set to **UPDATE**, only tuple data that meets the condition (the return value of **using_expression** is **TRUE**) can be updated. The operations that are affected include UPDATE, UPDATE ... RETURNING, and SELECT ... FOR UPDATE/SHARE.

  If **command** is set to **DELETE**, only tuple data that meets the condition (the return value of **using_expression** is **TRUE**) can be deleted. The operations that are affected include DELETE and DELETE ... RETURNING.

  The following table describes the relationship between row-level security policies and SQL statements.

  Table 7-219 Relationship between row-level security policies and SQL statements.

Command	SELECT/ALL Policy	UPDATE/ALL Policy	DELETE/ALL Policy
SELECT	Existing row	No	No
SELECT FOR UPDATE/SHARE	Existing row	Existing row	No
UPDATE	No	Existing row	No
UPDATE RETURNING	Existing row	Existing row	No
DELETE	No	No	Existing row
DELETE RETURNING	Existing row	No	Existing row

- **role_name**

  Specifies database users affected by a row-level security policy.

  CURRENT_USER indicates the username in the current operating environment. **SESSION_USER** indicates the session username. If this parameter is not specified, the default value **PUBLIC** is used, indicating that all database users are affected. You can specify multiple affected database users.

  NOTICE

  System administrators are not affected by row-level security.

- **using_expression**

  Specifies an expression defined for a row-level security policy (return type: Boolean).

  The expression cannot contain aggregate functions or window functions. In the statement rewriting phase of a query, if row-level security for a data table is enabled, the expressions that meet the specified conditions will be added to the plan tree. The expression is calculated for each tuple in the data table. For SELECT, UPDATE, and DELETE, row data is visible to the current user only when the return value of the expression is **TRUE**. If the expression returns **FALSE**, the tuple is invisible to the current user. In this case, the user cannot view the tuple through the SELECT statement, update the tuple through the UPDATE statement, or delete the tuple through the DELETE statement.

Examples

```
-- Create user alice.
gaussdb=# CREATE USER alice PASSWORD '********';

-- Create user bob.
gaussdb=# CREATE USER bob PASSWORD '********';

-- Create data table all_data.
gaussdb=# CREATE TABLE public.all_data(id int, role varchar(100), data varchar(100));

-- Insert data into the data table.
gaussdb=# INSERT INTO all_data VALUES(1, 'alice', 'alice data');
gaussdb=# INSERT INTO all_data VALUES(2, 'bob', 'bob data');
gaussdb=# INSERT INTO all_data VALUES(3, 'peter', 'peter data');

-- Grant the read permission on the all_data table to users alice and bob.
gaussdb=# GRANT SELECT ON all_data TO alice, bob;

-- Enable the row-level security policy.
gaussdb=# ALTER TABLE all_data ENABLE ROW LEVEL SECURITY;

-- Create a row-level security policy to specify that the current user can view only their own data.
gaussdb=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role = CURRENT_USER);

-- View information about the all_data table.
gaussdb=# \d+ all_data
                        Table "public.all_data"
 Column |          Type          | Modifiers | Storage  | Stats target | Description
--------+------------------------+-----------+----------+--------------+-------------
 id     | integer                |           | plain    |              |
 role   | character varying(100) |           | extended |              |
 data   | character varying(100) |           | extended |              |
Row Level Security Policies:
    POLICY "all_data_rls" FOR ALL
      TO public
      USING (((role)::name = "current_user"()))
Has OIDs: no
Options: orientation=row, compression=no, enable_rowsecurity=true

-- Run SELECT.
gaussdb=# SELECT * FROM all_data;
 id | role  |    data
----+-------+------------
  1 | alice | alice data
  2 | bob   | bob data
  3 | peter | peter data
(3 rows)

gaussdb=# EXPLAIN(COSTS OFF) SELECT * FROM all_data;
      QUERY PLAN
----------------------
```



```
 Seq Scan on all_data
(1 row)

-- Switch to user alice and run SELECT.
gaussdb=# SET ROLE alice PASSWORD '********';
gaussdb=> SELECT * FROM all_data;
 id | role  |    data
----+-------+------------
  1 | alice | alice data
(1 row)

gaussdb=> EXPLAIN(COSTS OFF) SELECT * FROM all_data;
 QUERY PLAN
------------------------------------------------------------------
 Seq Scan on all_data
   Filter: ((role)::name = 'alice'::name)
 Notice: This query is influenced by row level security feature
(3 rows)

-- Delete a row-level security policy.
gaussdb=> RESET ROLE;
gaussdb=# DROP ROW LEVEL SECURITY POLICY all_data_rls ON all_data;

-- Delete the all_data table.
gaussdb=# DROP TABLE public.all_data;

-- Delete users alice and bob.
gaussdb=# DROP USER alice, bob;
```

Helpful Links

DROP ROW LEVEL SECURITY POLICY and **ALTER ROW LEVEL SECURITY POLICY**

7.12.8.39 CREATE RULE

Description

Defines a new rewriting rule.

Precautions

- To define or modify rules for a table, you must be the owner of the table.
- If multiple rules of the same type are defined for the same table, the rules are triggered one by one by name in alphabetical order.
- In the view, the RETURNING clause can be added to the INSERT, UPDATE, and DELETE rules to return columns by view. If a rule is triggered by the **INSERT RETURNING**, **UPDATE RETURNING**, or **DELETE RETURNING** command, these clauses are used to calculate the output result. If a rule is triggered by a command without RETURNING, the RETURNING clause of the rule is ignored. Currently, only unconditional INSTEAD rules can contain the RETURNING clause, and only one RETURNING clause can exist in all rules of one event. This ensures that only one RETURNING clause can be used for result calculation. If the RETURNING clause does not exist in any valid rule, the RETURNING query in this view will be rejected.
- Currently, ON SELECT rules must be unconditional INSTEAD rules and must have actions consisting of a single SELECT query. Therefore, an ON SELECT rule actually turns a table into a view whose visible content is the content returned by the **SELECT** command of the rule, rather than the content in the table (if any).

Syntax

```
CREATE [ OR REPLACE ] RULE name AS ON event
    TO table_name [ WHERE condition ]
    DO [ ALSO | INSTEAD ] { NOTHING | command | ( command ; command ... ) };
```

Events include:

```
SELECT
INSERT
DELETE
UPDATE
```

Parameters

- name

  Name of the created rule. It must be unique among all the rules for the same table.

  Value range: a string, which complies with the **naming convention**. A value can contain a maximum of 63 characters.

- event

  One of the SELECT, INSERT, UPDATE, and DELETE events.

- table_name

  Name (optionally schema-qualified) of the table or view to which the rule applies.

- condition

  SQL condition expression that returns a Boolean value, which determines whether to execute the rule. Expressions cannot reference any table except **NEW** and **OLD**, and cannot have aggregate functions. You are advised not to use numeric types such as int as conditions, because such types can be implicitly converted to bool values (non-zero values are implicitly converted to **true** and **0** is implicitly converted to **false**), which may cause unexpected results.

- INSTEAD

  Specifies that a command is used to replace the initial event.

- ALSO

  Specifies that a command should be executed after the initial event. If neither **ALSO** nor **INSTEAD** is specified, **ALSO** is the default value.

- command

  Specifies the command that composes the rule action. A valid command is one of the SELECT, INSERT, UPDATE, and DELETE statements.

Examples

```
-- Create the tbl_rule1 and tbl_rule2 tables for creating rules.
gaussdb=# CREATE TABLE tbl_rule1(c1 int,c2 int,c3 int, c4 int);
gaussdb=# CREATE TABLE tbl_rule2(c1 int,c2 int);

-- Create rule rule_test and use ALSO to specify that commands are executed after the initial event is executed.
gaussdb=# CREATE RULE rule_test AS
    ON INSERT TO tbl_rule1
    DO ALSO INSERT INTO tbl_rule2 VALUES (new.c1, new.c2);

-- Insert data into the tbl_rule1 table and view data in the two tables.
```



```
gaussdb=# INSERT INTO tbl_rule1 VALUES(1,11,111,1111), (2,22,222,2222);
gaussdb=# SELECT * FROM tbl_rule1;
 c1 | c2 | c3  |  c4
----+----+-----+------
  1 | 11 | 111 | 1111
  2 | 22 | 222 | 2222
(2 rows)
gaussdb=# SELECT * FROM tbl_rule2;
 c1 | c2
----+----
  1 | 11
  2 | 22
(2 rows)

-- Drop the rule.
gaussdb=# DROP RULE rule_test ON tbl_rule1;

-- Drop the table.
gaussdb=# DROP TABLE tbl_rule1;
gaussdb=# DROP TABLE tbl_rule2;
```

NOTICE

- The name specified after ON SELECT rules must be "_RETURN".
- Currently, the ON SELECT rule must be INSTEAD SELECT, and the table specified by TO is converted to a view. The prerequisite is that the table is empty and does not have restrictions such as triggers, indexes, and child tables. That is, the table must be an initial empty table. Therefore, you are advised not to use ON SELECT rules. Instead, you can directly create a view.

7.12.8.40 CREATE SCHEMA

Description

Creates a schema.

Named objects are accessed either by "qualifying" their names with the schema name as a prefix, or by setting a search path that includes the desired schema. When creating named objects, you can also use the schema name as a prefix.

Optionally, CREATE SCHEMA can include sub-commands to create objects within the new schema. The sub-commands are treated essentially the same as separate commands issued after creating the schema. If the AUTHORIZATION clause is used, all the created objects are owned by this user.

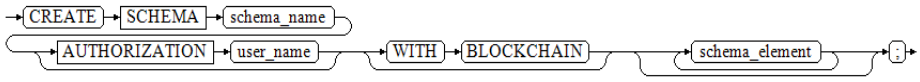
Precautions

- Only a user with the **CREATE** permission on the current database can perform this operation.
- The owner of an object created by a system administrator in a schema with the same name as a common user is the common user, not the system administrator.

Syntax

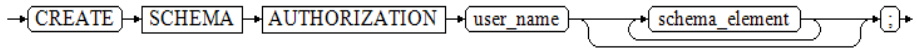
- Create a schema based on a specified name.


```
CREATE SCHEMA schema_name
    [ AUTHORIZATION user_name ] [WITH BLOCKCHAIN] [ schema_element [ ... ] ];
```



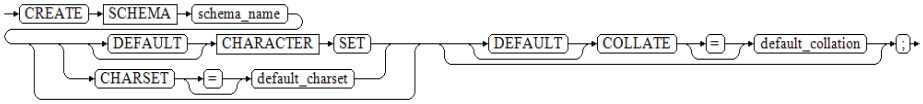
- Create a schema based on a username.

```
CREATE SCHEMA AUTHORIZATION user_name [ schema_element [ ... ] ];
```



- Create a schema and specify the default character set and collation.

```
CREATE SCHEMA schema_name
    [ [DEFAULT] CHARACTER SET | CHARSET [ = ] default_charset ] [ [ DEFAULT] COLLATE [ = ]
default_collation ];
```



Parameters

- **schema_name**

  Specifies the schema name.

  NOTICE

  - The schema name must be unique in the current database.
  - The schema name cannot be the same as the initial username of the current database.
  - The schema name cannot start with pg_.
  - The schema name cannot start with gs_role_.

  Value range: a string. It must comply with the **naming convention**.

- **AUTHORIZATION user_name**

  Specifies the owner of a schema. If **schema_name** is not specified, **user_name** will be used as the schema name. In this case, **user_name** can only be a role name.

  Value range: an existing username or role name

- **WITH BLOCKCHAIN**

  Specifies the tamper-proof attribute of a schema. In this mode, a row-store common user table is automatically extended to tamper-proof user table.

  NOTE

  To create a tamper-proof schema, set the GUC parameter **enable_ledger** to **on**. The default value is **off**, and the level is SIGHUP.

- **schema_element**

  Specifies an SQL statement defining an object to be created within the schema. Currently, only the CREATE TABLE, CREATE VIEW, CREATE INDEX, CREATE TABLE PARTITION, CREATE SEQUENCE, CREATE TRIGGER, and GRANT clauses are supported.

Objects created by sub-commands are owned by the user specified by AUTHORIZATION.

- **default_charset**

  Specifies the default character set of a schema. If this parameter is specified separately, the default collation of the schema is set to the default collation of the specified character set.

  This syntax is supported only when **sql_compatibility** is set to **'B'**.

- **default_collation**

  Specifies the default collation of a schema. If this parameter is specified separately, the default character set of the schema is set to the character set corresponding to the specified collation.

  This syntax is supported only when **sql_compatibility** is set to **'B'**. For details about the supported collation, see **Table 7-220**.

NOTE

If objects in the schema on the current search path are with the same name, specify the schemas for different objects. You can run **SHOW SEARCH_PATH** to check the schemas on the current search path.

Examples

```
-- Create and switch to the test database.
gaussdb=# CREATE DATABASE test1 WITH DBCOMPATIBILITY = 'B' ENCODING = 'UTF8' LC_COLLATE =
'zh_CN.utf8' LC_CTYPE = 'zh_CN.utf8';
gaussdb=# \c test1

-- Create the role1 role.
test1=# CREATE ROLE role1 IDENTIFIED BY '********';

-- Create a schema named role1 for the role1 role. The owner of the films and winners tables created by the clause is role1.
test1=# CREATE SCHEMA AUTHORIZATION role1
                CREATE TABLE films (title text, release date, awards text[])
                CREATE VIEW winners AS SELECT title, release FROM films WHERE awards IS NOT NULL;

-- Create a schema ds and set the default character set of the schema to utf8mb4 and the default collation to utf8mb4_bin. This syntax is supported only when sql_compatibility is set to 'B'.
test1=# CREATE SCHEMA ds CHARACTER SET utf8mb4 COLLATE utf8mb4_bin;

-- Delete the schema.
test1=# DROP SCHEMA role1 CASCADE;
test1=# DROP SCHEMA ds CASCADE;

-- Delete the user.
test1=# DROP USER role1 CASCADE;

-- Switch back to the initial database and delete the test database. Replace postgres with the actual database name.
test1=# \c postgres
gaussdb=# DROP DATABASE test1;
```

Helpful Links

ALTER SCHEMA and **DROP SCHEMA**

7.12.8.41 CREATE SECURITY LABEL

Description

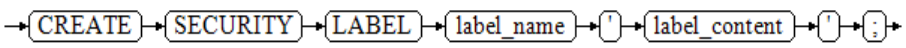
CREATE SECURITY LABEL creates a new security label for the specified security policy in the current database.

Precautions

An initial user, a user with the SYSADMIN permission, or a user who inherits the gs_role_seclabel permission of the built-in role can create security labels.

Syntax

```
CREATE SECURITY LABEL label_name 'label_content';
```



Parameters

- **label_name**

  Security label name, which must be unique in the database.

  Value range: a string of a maximum of 63 characters. Only lowercase letters (a to z), uppercase letters (A to Z), digits, underscores (_), and dollar signs ($) are allowed. If the length exceeds 63 characters, the database retains only the first 63 characters as the security label name.

- **label_content**

  Security label content. The requirements are as follows:

  A security label consists of only one level and at least one range, which are separated by a colon (:). The format is "level:range", for example, "L1:G2,G41,G6-G27".

  - There are 1024 levels named Li, where 1 ≤ i ≤ 1024. The levels meet a partial order relationship (if i ≤ j, then Li ≤ Lj). For example, L1 is lower than L3.
  - There are 1024 ranges named Gi, where 1 ≤ i ≤ 1024. You cannot compare sizes between ranges, but you can perform set operations. Multiple ranges are separated by commas (,), and a hyphen (-) is used to specify the interval. For example, {G2-G5} indicates {G2,G3,G4,G5}. {G1} is a subset of {G1, G6}.
  - The letters L and G must be capitalized and followed by at least one non-zero digit. Other characters are not allowed. In the {Gxxx-Gyyy} format, yyy must be greater than or equal to xxx.
  - If the input levels and ranges do not meet the requirements, the system reports an error.

CAUTION

Example:
gaussdb=# CREATE SECURITY LABEL sec_label3 'L3:'; // The label must contain at least one content range category.
ERROR: in label text "L3:", there at least have one level and one group

Examples

```
-- Create a security label sec_label.
gaussdb=# CREATE SECURITY LABEL sec_label 'L1:G4';

-- Create security label sec_label with the content of 'L1:G2,G4'.
gaussdb=# CREATE SECURITY LABEL sec_label 'L1:G2,G4';
ERROR:  security label "sec_label" already exists
-- Create security label sec_label1 with the content of 'L1:G2,G4'.
gaussdb=# CREATE SECURITY LABEL sec_label1 'L1:G2,G4';

-- Create security label sec_label2 with the content of 'L3:G1-G5'.
gaussdb=# CREATE SECURITY LABEL sec_label2 'L3:G1-G5';

-- View the security labels created in the system.
gaussdb=# SELECT * FROM gs_security_label;
 label_name | label_content
------------+---------------
 sec_label  | L1:G4
 sec_label1 | L1:G2,G4
 sec_label2 | L3:G1-G5
(3 rows)

-- Delete the existing security labels sec_label, sec_label1, and sec_label2.
gaussdb=# DROP SECURITY LABEL sec_label;
gaussdb=# DROP SECURITY LABEL sec_label1;
gaussdb=# DROP SECURITY LABEL sec_label2;

-- View the security labels created in the system again.
gaussdb=# SELECT * FROM gs_security_label;
 label_name | label_content
------------+---------------
(0 rows)
```

Helpful Links

DROP SECURITY LABEL and **SECURITY LABEL ON**

7.12.8.42 CREATE SEQUENCE

Description

CREATE SEQUENCE adds a sequence to the current database. The owner of a sequence is the user who creates the sequence.

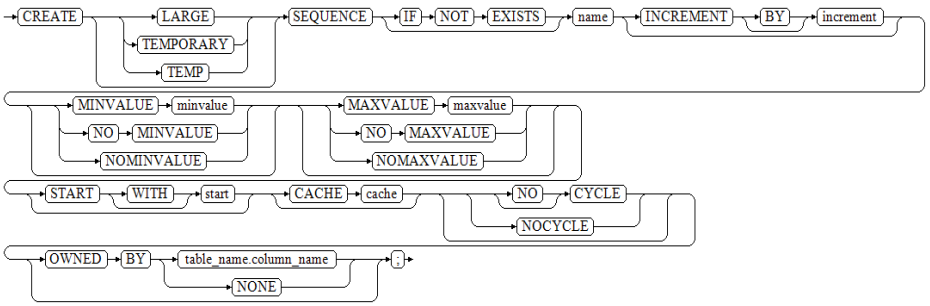
Precautions

- A sequence is a special table that stores arithmetic progressions. It has no actual meaning and is usually used to generate unique identifiers for rows or tables.
- If a schema name is given, the sequence is created in the specified schema; otherwise, it is created in the current schema. The sequence name must be different from the names of other sequences, tables, indexes, views in the same schema.
- After the sequence is created, functions **nextval()** and **generate_series(1,N)** insert data to the table. Make sure that the number of times for invoking **nextval** is greater than or equal to N+1. Otherwise, errors will be reported because the number of times for invoking function **generate_series()** is N+1.
- By default, the maximum value of **SEQUENCE** is 2^{63} – 1. If a large identifier is used, the maximum value can be 2^{127} – 1.

- A user granted with the CREATE ANY SEQUENCE permission can create sequences in the public and user schemas.

Syntax

```
CREATE [ LARGE | TEMPORARY | TEMP ] SEQUENCE [ IF NOT EXISTS ] name [ INCREMENT [ BY ] increment ]
    [ MINVALUE minvalue | NO MINVALUE | NOMINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE]
    [ START [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE | NOCYCLE ]
    [ OWNED BY { table_name.column_name | NONE } ];
```



Parameters

- **LARGE | TEMPORARY | TEMP**

  Specifies the keyword of a temporary sequence.

  NOTE

  - Enable the PostgreSQL compatibility mode for the database before creating a temporary sequence.
  - The lifecycle of a temporary sequence is at the session level, and temporary sequence objects are isolated by sessions. Sequences are automatically dropped on session exit.
  - Temporary sequences exist in a special schema. Each session has only one temporary schema. If a temporary schema is created in advance, the schema name can be provided when the temporary sequence is created. If a temporary schema is not created in advance, the schema name cannot be provided when the temporary sequence is created; in addition, each session can access only the objects in its own temporary schema and cannot access the objects in the temporary schemas of other sessions. If the accessed temporary schema does not belong to this session, an error is reported.
  - If temporary sequences exist, the existing permanent sequences with the same name (in this session) become invisible, but they can be referenced with a schema-limited name.

- **IF NOT EXISTS**

  When IF NOT EXISTS is specified, the system checks whether a relationship with the same name already exists in the current schema before creating a sequence. It is not created and a NOTICE is returned if a relationship with the same name already exists. When IF NOT EXISTS is not specified and a relationship with the same name exists in the schema, an ERROR is returned.

- **name**

  Specifies the name of a sequence to be created.

  Value range: a sting containing only lowercase letters, uppercase letters, special characters #_$, and digits.

- **increment**

  Optional. Specifies the step for a sequence. A positive number generates an ascending sequence, and a negative number generates a decreasing sequence.

  The default value is **1**.

  NOTE

  In B compatibility mode, if the step is a floating-point number, the value is automatically converted to an integer. In other modes, this parameter cannot be set to a floating-point number.

- **MINVALUE minvalue | NO MINVALUE| NOMINVALUE**

  Optional. Specifies the minimum value of the sequence. If **MINVALUE** is not declared, or **NO MINVALUE** is declared, the default value of the ascending sequence is **1**, and that of the descending sequence is **$-2^{63}-1$**. **NOMINVALUE** is equivalent to **NO MINVALUE**.

- **MAXVALUE maxvalue | NO MAXVALUE| NOMAXVALUE**

  Optional. Specifies the maximum value of the sequence. If **MAXVALUE** is not declared, or **NO MAXVALUE** is declared, the default value of the ascending sequence is **$2^{63}-1$**, and that of the descending sequence is **-1**. **NOMAXVALUE** is equivalent to **NO MAXVALUE**.

- **start**

  Optional. Specifies the start value of the sequence. The default value for an ascending sequence is **minvalue** and that for a descending sequence is **maxvalue**.

- **cache**

  Optional. Specifies the number of sequences stored in the memory for quick access purposes.

  Default value **1** indicates that one sequence can be generated each time.

  NOTE

  It is not recommended that you define **cache** and **maxvalue** or **minvalue** at the same time. The continuity of sequences cannot be ensured after **cache** is defined because unacknowledged sequences may be generated, causing waste of sequences.

- **[NO] CYCLE | NOCYCLE**

  Optional. Recycles sequences after the number of sequences reaches **maxvalue** or **minvalue**.

  If **NO CYCLE** is specified, any calling of nextval would return an error after the number of sequences reaches **maxvalue** or **minvalue**.

  - **NOCYCLE** is equivalent to **NO CYCLE**. The default value is **NO CYCLE**.
  - If **CYCLE** is specified, the sequence uniqueness cannot be ensured.

- **OWNED BY**

  Optional. Associates a sequence with a specified column included in a table. In this way, the sequence will be deleted when you delete its associated column or the table where the column belongs to. The associated table and sequence must be owned by the same user and in the same schema. OWNED BY only establishes the association between a table column and the sequence. Sequences on the column do not increase automatically when data is inserted.

The default value **OWNED BY NONE** indicates that such association does not exist.

NOTICE

You are advised not to use the sequence created using OWNED BY in other tables. If multiple tables need to share a sequence, the sequence must not belong to a specific table.

Examples

- Create an ascending sequence named **seq1**. The sequence starts from 101 and the step is 10.

```
gaussdb=# CREATE SEQUENCE seq1
    START 101
    INCREMENT 10;

-- Select the next number from the sequence.
gaussdb=# SELECT nextval('seq1');
 nextval
---------
     101
(1 row)
gaussdb=# SELECT nextval('seq1');
 nextval
---------
     111

-- Delete the sequence.
gaussdb=# DROP SEQUENCE seq1;
```

- Implement the auto-increment column.

```
-- Create a table.
gaussdb=# CREATE TABLE test1(id int PRIMARY KEY, name varchar(20));

-- Create a sequence joined with the table.
gaussdb=# CREATE SEQUENCE test_seq2
    START 1
    NO CYCLE
    OWNED BY test1.id;

-- Set the default value of a column.
gaussdb=# ALTER TABLE test1 ALTER COLUMN id SET DEFAULT nextval('test_seq2'::regclass);

-- Insert data.
gaussdb=# INSERT INTO test1 (name) values ('Joe'),('Scott'),('Ben');

-- Query data in the table.
gaussdb=# SELECT * FROM test1;
 id | name
----+-------
  1 | Joe
  2 | Scott
  3 | Ben
(3 rows)

-- Delete the sequence and the table.
gaussdb=# DROP SEQUENCE test_seq2 CASCADE;
gaussdb=# DROP TABLE test1;
```

Helpful Links

DROP SEQUENCE and **ALTER SEQUENCE**

7.12.8.43 CREATE SERVER

Description

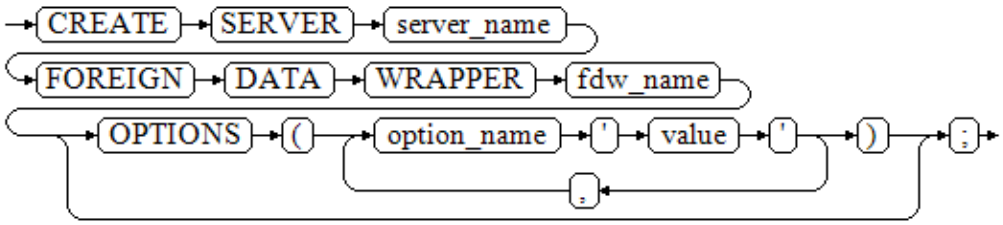
Defines a new foreign server.

Precautions

When multi-layer quotation marks are used for sensitive columns (such as **password** and **secret_access_key**) in OPTIONS, the semantics is different from that in the scenario where quotation marks are not used. Therefore, sensitive columns are not identified for anonymization.

Syntax

```
CREATE SERVER server_name
    FOREIGN DATA WRAPPER fdw_name
    [ OPTIONS ( { option_name ' value ' } [, ...] ) ];
```



Parameters

- **server_name**

  Specifies the server name.

  Value range: a string containing no more than 63 bytes.

- **fdw_name**

  Specifies the name of the foreign data wrapper.

  Value range: **dist_fdw**, **log_fdw**, and **file_fdw**. **log_fdw** and **file_fdw** are used only for syntax compatibility in centralized mode. They can be used to create foreign tables but are not actually used.

- **OPTIONS ({ option_name ' value ' } [, ...])**

  Specifies options for the server. These options typically define the connection details of the server, but the actual names and values depend on the foreign data wrapper of the server.

  – Specifies the parameters for the foreign server. The detailed parameter description is as follows:

    ▪ encrypt

      Specifies whether data is encrypted. This parameter is available only when **type** is **OBS**. The default value is **on**.

      Value range:

      ○ **on** indicates that data is encrypted and HTTPS is used for communication.

    - **off** indicates that data is not encrypted and HTTP is used for communication.

  - access_key

    Specifies the AK (obtained by users from the OBS console) used for the OBS access protocol. This parameter is available only when **type** is set to **OBS**.

  - secret_access_key

    Specifies the SK value (obtained by users from the OBS console) used for the OBS access protocol. This parameter is available only when **type** is set to **OBS**.

In addition to the connection parameters supported by libpq, the following parameters are provided:

- **fdw_startup_cost**

  Estimates the startup time required for a foreign table scan. This value usually contains the time taken to establish a connection, analyze the request at the remote end, and generate a plan. The default value is **100**. The value is a real number greater than 0.

- **fdw_typle_cost**

  Specifies the additional consumption when each tuple is scanned on a remote server. The value specifies the extra consumption of data transmission between servers. The default value is **0.01**. The value is a real number greater than 0.

Examples

```
-- Create a server.
gaussdb=# CREATE SERVER my_server FOREIGN DATA WRAPPER log_fdw;
CREATE SERVER

-- Delete the server.
gaussdb=# DROP SERVER my_server;
DROP SERVER
```

Helpful Links

ALTER SERVER and **DROP SERVER**

7.12.8.44 CREATE SYNONYM

Description

Creates a synonym object. A synonym is an alias of a database object and is used to record the mapping between database object names. You can use synonyms to access associated database objects.

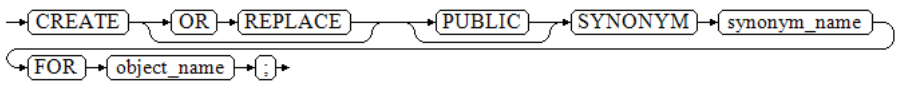
Precautions

- The user who creates the synonym will be the owner of the synonym.
- If the schema name is specified, create a synonym in the specified schema. Otherwise, a synonym is created in the current schema.

- Database objects that can be accessed using synonyms include tables, views, types, packages, functions, stored procedures, sequences, or other synonyms.
- To use synonyms, you must have the required permissions on associated objects.
- The following DML statements support synonyms: SELECT, INSERT, UPDATE, DELETE, EXPLAIN, and CALL.
- The CREATE SYNONYM statement of an associated function or stored procedure cannot be used in a stored procedure. You are advised to use synonyms existing in the pg_synonym system catalog in the stored procedure.
- You are advised not to create synonyms for temporary tables. To create a synonym, you need to specify the schema name of the target temporary table. Otherwise, the synonym cannot be used normally. In addition, you need to run the **DROP SYNONYM** command before the current session ends.
- After an original object is deleted, the synonym associated with the object will not be deleted in cascading mode. If you continue to access the synonym, for accessed tables, a message is displayed indicating that the synonym has expired; for accessed functions, stored procedures, and packages, a message is displayed indicating that the objects do not exist.
- Users granted the CREATE ANY SYNONYM permission can create synonyms in user schemas.
- Synonyms cannot be created for encrypted tables that contain encrypted columns and views, functions, and stored procedures based on encrypted tables.
- A synonym can be associated with a package, but not with a function in a package. You can use synonyms associated with a package to access functions and variables in the package.
- If the schema of a synonym is the schema to which the user belongs, the owner of the synonym is also the owner of the schema. In other scenarios, the owner of the synonym is the creator of the synonym by default.
- If **SEARCH_PATH** is set and no synonym schema is specified, for stored procedures and functions, the PG_PROC table is preferentially searched by name. If no function with the same name exists, synonyms are searched based on **SEARCH_PATH**. For other objects, **SEARCH_PATH** is preferentially searched, if their schemas are the same as that of synonyms, the objects are accessed prior to synonyms.
- Objects associated with synonyms cannot be accessed using DDL statements, such as CREATE, DROP, and ALTER.
- Nested synonyms are supported. When you search for a synonym, if the synonym is associated with another synonym, the system continues to search for the associated synonym until the last associated object is found.
- You cannot use \d, \df, or \sf to access information about associated objects through synonyms.

Syntax

```
CREATE [ OR REPLACE ] [PUBLIC] SYNONYM synonym_name
    FOR object_name;
```



Parameters

- **OR REPLACE**

  (Optional) Redefines the synonym if it already exists.

- **PUBLIC**

  (Optional) Creates a PUBLIC synonym.

  NOTE

  - PUBLIC synonyms must be unique in the same database.
  - If the database is upgraded from a version that does not support PUBLIC synonyms to a version that supports PUBLIC synonyms, PUBLIC synonyms cannot be created or deleted before the upgrade is committed.
  - All users can access PUBLIC synonyms. Except the initial users and system administrators, you must have the CREATE PUBLIC SYNONYM and DROP PUBLIC SYNONYM permissions to create and delete PUBLIC synonyms, respectively.
  - For the PUBLIC synonym, the values of **synnamespace** and **synowner** in the PG_SYNONYM system catalog are **0**, the value of **owner** in the ADM_SYNONYMS and DB_SYNONYMS system views is **PUBLIC**, and the value of **schema_name** is **NULL**.
  - If no synonym schema is specified, the system searches for objects with the same name to determine whether the objects exist. Then, the system searches for synonyms by **SEARCH_PATH**. Finally, the system searches for PUBLIC synonyms. If a synonym schema is specified, PUBLIC synonyms are not retrieved.

- **synonym_name**

  Specifies the name of the synonym to be created, which can contain the schema name.

  Value range: a string. It must comply with the **naming convention**.

- **object_name**

  Specifies the name of an object that is associated (optionally with schema names).

  Value range: a string. It must comply with the **naming convention**.

  NOTE

  - **object_name** can be the name of an object that does not exist.
  - **object_name** can be the name of a remote object accessed by using a database link. For details about how to use database links, see **DATABASE LINK**.

CAUTION

Do not create aliases for functions that contain passwords and other sensitive information, such as the encryption functions gs_encrypt and gs_encrypt_bytera, and the decryption functions gs_decrypt and gs_decrypt_bytea or use aliases to call the functions to prevent sensitive information leakage.

Examples

```
-- Create schema ot.
gaussdb=# CREATE SCHEMA ot;
```



```
-- Create a table ot.test_tbl1.
gaussdb=# CREATE TABLE ot.test_tbl1(c1 INT, c2 INT);
gaussdb=# INSERT INTO ot.test_tbl1 values(1,1);

-- View the current value of search_path.
gaussdb=# SHOW search_path;
  search_path
----------------
 "$user",public
(1 row)

-- The current value of search_path does not contain ot, and the current user is not ot. Therefore, an error is reported when you directly view the table name.
gaussdb=# SELECT * FROM test_tbl1;
ERROR:  relation "test_tbl1" does not exist
LINE 1: SELECT * FROM test_tbl1;

-- Create a synonym.
gaussdb=# CREATE OR REPLACE SYNONYM test_tbl1 FOR ot.test_tbl1;
-- Use the synonym.
gaussdb=# SELECT * FROM test_tbl1;
 c1 | c2
----+----
  1 |  1
(1 row)
gaussdb=# INSERT INTO test_tbl1 VALUES (2,2);

-- Delete.
gaussdb=# DROP SYNONYM test_tbl1;
gaussdb=# DROP TABLE ot.test_tbl1;
gaussdb=# DROP SCHEMA ot CASCADE;
```

Helpful Links

ALTER SYNONYM and **DROP SYNONYM**

7.12.8.45 CREATE TABLE

Description

Creates an initially empty table in the current database. The table will be owned by the creator. Row-store tables are created by default.

Precautions

- If an error occurs during table creation, after it is fixed, the system may fail to delete the empty disk files created before the last automatic clearance. This problem seldom occurs and does not affect system running of the database.
- When JDBC is used, the **DEFAULT** value can be set through **PrepareStatement**.
- A user granted with the CREATE ANY TABLE permission can create tables in the public and user schemas. To create a table that contains serial columns, you must also obtain the CREATE ANY SEQUENCE permission to create sequences.
- The XML type cannot be used as a primary key or foreign key.
- If you add a row-level expression when adding or changing an ILM policy for a data object, note that the row-level expression supports only the functions listed in the whitelist. For details about the whitelist function list, see **Row Expression Function Whitelist**.

- The number of table constraints cannot exceed 32,767.

Syntax

Create a table.

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT EXISTS ] table_name
   {({ column_name data_type [ CHARACTER SET | CHARSET charset ] [ compress_mode ] [ COLLATE
collation ] [ column_constraint [ ... ] ]
       | table_constraint
       | LIKE source_table [ like_option [...] ] }
       [, ... ])
   | LIKE source_table }
   [ table_option [ [ , ] ... ] ]
   [ WITH ( {storage_parameter = value} [, ... ] ) ]
   [ ON COMMIT { PRESERVE ROWS | DELETE ROWS } ]    [ COMPRESS | NOCOMPRESS ]
   [ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF
{ NO MODIFICATION } [ ON ( EXPR )]]
   [ TABLESPACE tablespace_name ];
```

- **table_option** is as follows:

  ```
  { COMMENT [ = ] 'string' |
    AUTO_INCREMENT [ = ] value |
    [ DEFAULT ] CHARACTER SET | CHARSET [ = ] default_charset |
    [ DEFAULT ] COLLATE [ = ] default_collation |
    ENGINE [ = ] { InnoDB | 'InnoDB' | "InnoDB" } }
  ```

- **column_constraint** is as follows:

  ```
  [ CONSTRAINT constraint_name ]
  { NOT NULL |
    NULL |
    CHECK ( expression ) |
    DEFAULT default_expr |
    ON UPDATE update_expr |
    GENERATED ALWAYS AS ( generation_expr ) [STORED] |
    GENERATED [ ALWAYS | BY DEFAULT [ ON NULL ] ] AS IDENTITY [ ( identity_options ) ] |
    AUTO_INCREMENT |
    COMMENT 'string' |
    UNIQUE [KEY] index_parameters |
    PRIMARY KEY index_parameters |
    ENCRYPTED WITH ( COLUMN_ENCRYPTION_KEY = column_encryption_key, ENCRYPTION_TYPE =
  encryption_type_value ) |
    REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
       [ ON DELETE action ] [ ON UPDATE action ] }
  [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
  ```

- **compress_mode** of a column is as follows:

  ```
  { DELTA | PREFIX | DICTIONARY | NUMSTR | NOCOMPRESS }
  ```

- **table_constraint** is as follows:

  ```
  [ CONSTRAINT [ constraint_name ] ]
  { CHECK ( expression ) |
    UNIQUE [ index_name ] [ USING method ] ( { { column_name [ ( length ) ] | ( expression ) } [ ASC |
  DESC ] } [, ... ] ) index_parameters |
    PRIMARY KEY [ USING method ] ( { column_name [ ASC | DESC ] } [, ... ] ) index_parameters
    |
    FOREIGN KEY [ index_name ] ( column_name [, ... ] ) REFERENCES reftable [ ( refcolumn [, ... ] ) ]
       [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ] [ ON UPDATE
  action ] }
  [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
  { [ COMMENT 'string' ] [ ... ] }
  ```

- **like_option** is as follows:

  ```
  { INCLUDING | EXCLUDING } { DEFAULTS | GENERATED | CONSTRAINTS | INDEXES | STORAGE |
  COMMENTS | PARTITION | RELOPTIONS | UPDATE | IDENTITY | ALL }
  ```

- **index_parameters** is as follows:

  ```
  [ WITH ( {storage_parameter = value} [, ... ] ) ]
  [ USING INDEX TABLESPACE tablespace_name ]
  ```

- **update_expr** is as follows:


```
{ CURRENT_TIMESTAMP | LOCALTIMESTAMP | NOW() }
```

- **identity_options** is as follows:

```
[ INCREMENT [ BY ] increment ] [ MINVALUE minvalue | NO MINVALUE | NOMINVALUE]
[ MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE] [ START [ WITH ] start ] [ CACHE cache |
NOCACHE ] [ [ NO ] CYCLE | NOCYCLE] [ SCALE [ EXTEND | NO EXTEND] | NOSCALE ]
```

Parameters

- **UNLOGGED**

  If this keyword is specified, the created table is an unlogged table. Data written to unlogged tables is not written to the WALs, which makes them considerably faster than ordinary tables. However, an unlogged table is automatically truncated after conflicts, OS restart, database restart, primary/standby switchover, power-off, or abnormal shutdown, incurring data loss risks. Contents of an unlogged table are also not replicated to standby nodes. Any indexes created on an unlogged table are not automatically logged as well.

  Usage scenario: Unlogged tables do not ensure data security. Users can back up data before using unlogged tables; for example, users should back up the data before a system upgrade.

  Troubleshooting: If data is missing in the indexes of unlogged tables due to some unexpected operations such as an unclean shutdown, users should rebuild the indexes with errors.

- **GLOBAL | LOCAL**

  When creating a temporary table, you can specify the **GLOBAL** or **LOCAL** keyword before **TEMP** or **TEMPORARY**. If the keyword **GLOBAL** is specified, GaussDB creates a global temporary table. Otherwise, GaussDB creates a local temporary table.

- **TEMPORARY | TEMP**

  If **TEMP** or **TEMPORARY** is specified, the created table is a temporary table. Temporary tables are classified into global temporary tables and local temporary tables. If the keyword **GLOBAL** is specified when a temporary table is created, the table is a global temporary table. Otherwise, the table is a local temporary table.

  The metadata of the global temporary table is visible to all sessions. After the sessions end, the metadata still exists. The user data, indexes, and statistics of a session are isolated from those of another session. Each session can only view and modify the data committed by itself. Global temporary tables have two schemas: **ON COMMIT PRESERVE ROWS** and **ON COMMIT PRESERVE ROWS**. In session-based **ON COMMIT PRESERVE ROWS** schema, user data is automatically cleared when a session ends. In transaction-based **ON COMMIT DELETE ROWS** schema, user data is automatically cleared when the commit or rollback operation is performed. If the **ON COMMIT** option is not specified during table creation, the session level is used by default. Different from local temporary tables, you can specify a schema that does not start with **pg_temp_** when creating a global temporary table.

  A local temporary table is automatically dropped at the end of the current session. Therefore, you can create and use temporary tables in the current session as long as the connected database node in the session is normal. Temporary tables are created only in the current session. If a DDL statement involves operations on temporary tables, a DDL error will be generated.

Therefore, you are advised not to perform operations on temporary tables in DDL statements. **TEMP** is equivalent to **TEMPORARY**.

NOTICE

- Local temporary tables are visible to the current session through the schema starting with **pg_temp**. Users should not delete schemas starting with **pg_temp** or **pg_toast_temp**.
- If **TEMPORARY** or **TEMP** is not specified when you create a table but its schema is set to that starting with **pg_temp_** in the current session, the table will be created as a temporary table.
- If global temporary tables and indexes are being used by other sessions, do not perform **ALTER** or **DROP** (except the **ALTER INDEX index_name REBUILD** command).
- The DDL of a global temporary table affects only the user data and indexes of the current session. For example, TRUNCATE, REINDEX, and ANALYZE are valid only for the current session.
- You can set the GUC parameter **max_active_global_temporary_table** to determine whether to enable the global temporary table function. If **max_active_global_temporary_table** is set to **0**, the global temporary table function is disabled.
- A temporary table is visible only to the current session. Therefore, it cannot be used together with **\parallel on**.
- Temporary tables do not support primary/standby switchover.
- The global temporary table does not respond to automatic clearance. In persistent connection scenarios, you are advised to use the global temporary table in the ON COMMIT DELETE ROWS clause or periodically and manually execute the VACUUM statement. Otherwise, Clogs may not be reclaimed.
- Global temporary tables do not support the following scenarios:
  - Global temporary sequences cannot be created. Global temporary tables of each session use shared sequences, which can only ensure uniqueness but not continuity.
  - Global temporary views cannot be created.
  - Partitioned tables cannot be created.
  - Hash bucket tables cannot be created.
  - Extended statistics are not supported.

- **IF NOT EXISTS**

  Sends a notice, but does not throw an error, if a table with the same name exists.

- **table_name**

  Specifies the name of the table to be created.

NOTICE

Some processing logic of materialized views determines whether a table is the log table of a materialized view or a table associated with a materialized view based on the table name prefix. Therefore, do not create a table whose name prefix is **mlog_** or **matviewmap_**. Otherwise, some functions of the table are affected.

- **column_name**

  Specifies the name of a column to be created in the new table.

- **constraint_name**

  Specifies the name of the constraint specified during table creation.

NOTICE

The constraint_name is optional in B-compatible mode (**sql_compatibility = 'B'**). For other modes, constraint_name must be added.

- **index_name**

  Index name

NOTICE

- The **index_name** is supported only in B-compatible databases (that is, **sql_compatibility** set to **'B'**).
- For foreign key constraints, if constraint_name and index_name are specified at the same time, constraint_name is used as the index name.
- For a unique key constraint, if both constraint_name and index_name are specified, index_name is used as the index name.

- **USING method**

  Specifies the name of the index method to be used.

  For details about the value range, see USING method in **Parameters**.

NOTICE

- The USING method is supported only in B-compatible databases (that is, **sql_compatibility** set to **'B'**).
- In B-compatible mode, if USING method is not specified, the default index method is btree for Astore or UB-tree for Ustore.
- If the storage mode of a table is Ustore and the constraint in the SQL statement is specified as USING BTREE, the underlying layer automatically creates the constraint as USING UBTREE.

- **ASC | DESC**

  ASC specifies an ascending (default) sort order. **DESC** specifies a descending sort order.

NOTICE

The ASC|DESC is supported only in B-compatible databases (that is, **sql_compatibility** set to **'B'**).

- **expression**

  Specifies an expression index constraint based on one or more columns of the table. It must be written in parentheses.

NOTICE

Expression indexes in the UNIQUE constraint are supported only in B-compatible databases (that is, **sql_compatibility** set to **'B'**).

- **data_type**

  Specifies the data type of the column.

- **compress_mode**

  Specifies whether to compress a table column. The option specifies the algorithm preferentially used by table columns. Row-store tables do not support compression.

  Value range: **DELTA**, **PREFIX**, **DICTIONARY**, **NUMSTR**, and **NOCOMPRESS**

- **CHARACTER SET | CHARSET charset**

  Specifies the character set of a table column. If this parameter is specified separately, the collation of the table column is set to the default collation of the specified character set.

  This syntax is supported only in B-compatible databases (that is, **sql_compatibility** set to **'B'**).

- **COLLATE collation**

  Assigns a collation to the column (which must be of a collatable data type). If no collation is specified, the default collation is used. You can run the **SELECT * FROM pg_collation;** command to query collation rules from the pg_collation system catalog. The default collation rule is the row starting with **default** in the query result. In B-compatible databases (that is, **sql_compatibility = 'B'**), **utf8mb4_bin**, **utf8mb4_general_ci**, **utf8mb4_unicode_ci**, **binary**, **gbk_chinese_ci**, **gbk_bin**, **gb18030_chinese_ci**, and **gb18030_bin** are also supported.

  NOTE

  - Only the character type supports the specified character set. If the binary character set or collation is specified, the character type is converted to the corresponding binary type. If the type mapping does not exist, an error is reported. Currently, only the mapping from the TEXT type to the BLOB type is available.
  - Except the binary character set and collation, only the character set that is the same as the database encoding can be specified.
  - If the character set or collation of a column is not explicitly specified and the default character set or collation of the table is specified, the character set or collation of the column is inherited from the table. If the default character set or collation of a table does not exist, the character set and collation of table columns inherit the character set and collation of the current database when **b_format_behavior_compat_options** contains **'default_collation'**.

Table 7-220 Character sets and collation supported in mode B (sql_compatibility = 'B')

Collation	Character Set	Description
utf8mb4_general_ci	utf8mb4 (utf8)	The general collation is used, which is case insensitive.
utf8mb4_unicode_ci	utf8mb4 (utf8)	The general collation is used, which is case insensitive.
utf8mb4_bin	utf8mb4 (utf8)	The binary collation is used, which is case sensitive.
binary	binary	The binary collation is used.
gbk_chinese_ci	gbk	The Chinese collation is used.
gbk_bin	gbk	The binary collation is used, which is case sensitive.
gb18030_chinese_ci	gb18030	The Chinese collation is used.
gb18030_bin	gb18030	The binary collation is used, which is case sensitive.
utf8mb4_0900_ai_ci	utf8mb4	The unicode collation algorithm (UCA) rule is used, which is case-insensitive.
utf8_general_ci	utf8	The general collation is used, which is case insensitive.
utf8_bin	utf8	The binary collation is used, which is case sensitive.

- **LIKE source_table [like_option ...]**

  Specifies a table from which the new table automatically copies all column names, their data types, and their not-null constraints.

  The new table and the original table are decoupled after creation is complete. Changes to the original table will not be applied to the new table, and it is not possible to include data of the new table in scans of the original table.

  The copied columns and constraints are not merged with similarly named columns and constraints. If the same name is specified explicitly or in another **LIKE** clause, an error is reported.

  - The default expressions are copied from the original table to the new table only if **INCLUDING DEFAULTS** is specified. The default behavior is to exclude default expressions, resulting in the copied columns in the new table having default values **NULL**.
  - If **INCLUDING UPDATE** is specified, the **ON UPDATE CURRENT_TIMESTAMP** attribute of the source table column is copied to the new table column. By default, the generated expression is not copied.

- If **INCLUDING GENERATED** is specified, the generated expression of the source table column is copied to the new table. By default, the generated expression is not copied.
- The **CHECK** constraints are copied from the original table to the new table only when **INCLUDING CONSTRAINTS** is specified. Other types of constraints are never copied to the new table. Not-null constraints are always copied to the new table. These rules also apply to column constraints and table constraints.
- Any indexes on the original table will not be created on the new table, unless the **INCLUDING INDEXES** clause is specified.
- **STORAGE** settings for the source column definitions are copied only if **INCLUDING STORAGE** is specified. The default behavior is to exclude **STORAGE** settings.
- If **INCLUDING COMMENTS** is specified, comments for the copied columns, constraints, and indexes are copied. The default behavior is to exclude comments.
- If **INCLUDING PARTITION** is specified, the partition definitions of the source table are copied to the new table, and the new table no longer uses the **PARTITION BY** clause. By default, the partition definition of the source table is not copied. If the source table has an index, you can use the **INCLUDING PARTITION INCLUDING INDEXES** syntax. If only **INCLUDING INDEXES** is used for a partitioned table, the target table will be defined as an ordinary table, but the index is a partitioned index. In this case, an error will be reported because ordinary tables do not support partitioned indexes.
- If **INCLUDING RELOPTIONS** is specified, the new table will copy the storage parameter (that is, **WITH** clause) of the source table. The default behavior is to exclude partition definition of the storage parameter of the original table.
- If **INCLUDING IDENTITY** is specified, the identity of the source table will be copied to the new table, and a SEQUENCE with the same **SEQUENCE** parameter as that of the source table will be created. By default, the identity of the source table is not copied.
- INCLUDING ALL contains INCLUDING DEFAULTS, INCLUDING UPDATE, INCLUDING GENERATED, INCLUDING CONSTRAINTS, INCLUDING INDEXES, INCLUDING STORAGE, INCLUDING COMMENTS, INCLUDING PARTITION, INCLUDING RELOPTIONS, and INCLUDING IDENTITY.

NOTICE

- If the original table contains a sequence with the SERIAL, BIGSERIAL, SMALLSERIAL, or LARGESERIAL data type, or a column in the original table is a sequence by default and the sequence is created for this table (created by using CREATE SEQUENCE ... OWNED BY), these sequences will not be copied to the new table, and another sequence specific to the new table will be created. This is different from earlier versions. To share a sequence between the source table and new table, create a shared sequence (do not use **OWNED BY**) and set a column in the source table to this sequence.
- You are advised not to set a column in the source table to the sequence specific to another table especially when the table is distributed in specific node groups, because doing so may result in CREATE TABLE ... LIKE execution failures. In addition, doing so may cause the sequence to become invalid in the source sequence because the sequence will also be deleted from the source table when it is deleted from the table that the sequence is specific to. To share a sequence among multiple tables, you are advised to create a shared sequence for them.
- **EXCLUDING** of a partitioned table must be used together with **INCLUDING ALL**, for example, **INCLUDING ALL EXCLUDING DEFAULTS**, except for **DEFAULTS** of the source partitioned table.
- The CREATE TABLE table_name LIKE source_table syntax is supported only when **sql_compatibility** is set to **'B'** (B-compatible database), **b_format_version** is set to **5.7**, and **b_format_dev_version** is set to **s2**.
- In a B-compatible database, if **b_format_version** is set to **5.7** and **b_format_dev_version** is set to **s2**, **INCLUDING** and **EXCLUDING** cannot be specified. In this case, it is equivalent to specify **INCLUDING ALL** by default.

- **AUTO_INCREMENT [=] value**

  This clause specifies an initial value for an auto-increment column. The value must be a positive number and cannot exceed $2^{127} - 1$.

NOTICE

This clause takes effect only when **sql_compatibility** is set to **'B'**.

- **COMMENT [=] 'string'**
  - The COMMENT [=] 'string' clause is used to add comments to a table.
  - The COMMENT 'string' in column_constraint indicates that comments are added to a column.
  - The COMMENT 'string' in table_constraint indicates that comments are added to the indexes corresponding to the primary key and unique key.

NOTICE

- This clause takes effect only when **sql_compatibility** is set to **'B'**.
- A table-level comment can contain a maximum of 2048 characters, and a column-level or index-level comment can contain a maximum of 1024 characters.
- Comments in table_constraint support only primary keys and unique keys.

- **ENGINE**

  Supported in B-compatible mode and used only for syntax adaptation. Only InnoDB can be set and no actual effect is achieved.

NOTICE

The ENGINE syntax cannot be used in the CREATE TABLE table_name LIKE source_table syntax.

- **WITH ({ storage_parameter = value } [, ...])**

  Specifies an optional storage parameter for a table or an index. The WITH clause used for tables can also contain **OIDS=FALSE** to specify that rows of the new table should not contain OIDs.

  NOTE

  When using **Numeric** of any precision to define a column, specify precision **p** and scale **s**. When precision and scale are not specified, the input will be displayed.

  The description of parameters is as follows:

  – FILLFACTOR

    The fill factor of a table is a percentage from 10 to 100. If the Ustore is used, the default value is **92**. If the Astore is used, the default value is **100** (completely filled). When a smaller fill factor is specified, **INSERT** operations fill table pages only to the indicated percentage. The remaining space on each page is reserved for updating rows on that page. This gives **UPDATE** a chance to place the updated copy of a row on the same page, which is more efficient than placing it on a different page. For a table whose entries are never updated, setting the fill factor to **100** (complete filling) is the best choice, but in heavily updated tables a smaller fill factor would be appropriate.

    Value range: 10–100

  – ORIENTATION

    Specifies the storage mode of table data. This parameter can be set only once.

    Value range:

    ▪ **ROW** indicates that table data is stored in rows.

      ROW applies to OLTP service and scenarios with a large number of point queries or addition/deletion operations.

    Default value:

    If an ordinary tablespace is specified, the default is **ROW**.

- STORAGE_TYPE

  Specifies the storage engine type. This parameter cannot be modified once it is set.

  Value range:

  - **USTORE** indicates that tables support the in-place update storage engine. Note that the **track_counts** and **track_activities** parameters must be enabled when the Ustore table is used. Otherwise, space bloating may occur.
  - **ASTORE** indicates that tables support the append-only storage engine.

  Default value:

  If **ORIENTATION** and **STORAGE_TYPE** are not specified, Ustore is used by default, indicating that the table supports the in-place update-based storage engine.

- INIT_TD

  Specifies the number of TDs to be initialized when a Ustore table is created. This parameter can be modified by running the **ALTER TABLE** command. Note that this parameter affects the maximum size of a single tuple stored on the data page. The conversion method is MAX_TUPLE_SIZE = BLCKSZ - INIT_TD * TD_SIZE. For example, if you change the number of INIT_TD from 4 to 8, the maximum size of a single tuple decreases by 4 x INIT_TD.

  Value range: 2 to 128

  Default value: **4**

- COMPRESSION

  Specifies the compression level of table data. It determines the compression ratio and time. Generally, the higher the level of compression, the higher the ratio, the longer the time; and the lower the level of compression, the lower the ratio, the shorter the time. The actual compression ratio depends on the distribution mode of table data loaded. Row-store tables do not support compression.

  Value range: For row-store tables, the valid values are **YES** and **NO**.

  Default value: **NO**

- COMPRESSLEVEL

  Specifies the table data compression ratio and duration at the same compression level. This divides a compression level into sublevels, providing more choices for compression ratio and duration. As the value becomes greater, the compression ratio becomes higher and duration longer at the same compression level.

  Value range: 0 to 3

  Default value: **0**

- segment

  The data is stored in segment-page mode. This parameter supports only row-store tables. Column-store tables are not supported. Protection against unauthorized deletion and damage of physical files 1 to 5 is not supported.

Value range: **on** and **off**

Default value: **off**

- enable_tde

  Indicates that the table is an encrypted table. The database automatically encrypts the data in the encryption table before storing it. Before using this parameter, ensure that TDE has been enabled using the GUC parameter **enable_tde** and the information for accessing the key service has been set using the GUC parameter **tde_key_info**. For details about how to use this parameter, see section "Transparent Data Encryption" in *Feature Guide*. This parameter applies only to row-store tables, segment-page tables, temporary tables, and unlogged tables.

  Value range: **on** and **off** When **enable_tde** is set to **on**, **key_type**, **tde_cmk_id**, and **dek_cipher** are automatically generated by the database and cannot be manually specified or modified.

  Default value: **off**

- encrypt_algo

  Specifies the encryption algorithm for an encrypted table. This parameter must be used together with **enable_tde**.

  Value range: a string. The value can be **AES_128_CTR** or **SM4_CTR**.

  Default value: null if **enable_tde** and **AES_128_CTR** if **enable_tde** is set.

- dek_cipher

  Specifies the DEK ciphertext. After a user sets the **enable_tde** parameter for a table, the database automatically generates a data key.

  Value range: a string

  Default value: null

- key_type

  Specifies the type of the master key. After the **enable_tde** parameter is set for a table, the database automatically obtains the master key type from the GUC parameter **tde_key_info**.

  Value range: a string

  Default value: null

- cmk_id

  Specifies the ID of the master key. After the **enable_tde** parameter is set for a table, the database automatically obtains the master key ID from the GUC parameter **tde_key_info**.

  Value range: a string

  Default value: null

- parallel_workers

  Number of bgworker threads started when an index is created. For example, value **2** indicates that two bgworker threads are started to create indexes concurrently.

  Value range: [0,32], int type. The value **0** indicates that concurrent index creation is disabled.

  Default value: If this parameter is not set, the concurrent index creation function is disabled.

  - hasuids

    If this parameter is set to **on**, a unique table-level ID is allocated to a tuple when the tuple is updated.

    Value range: **on** and **off**

    Default value: **off**

  - collate

    In B-compatible databases (**sql_compatibility** = **'B'**), this parameter is used to record the default collation of tables. Generally, this parameter is used only for internal storage, import, and export. You are advised not to specify or modify this parameter.

    Value range: OIDs in the collation that is independently supported in B-compatible databases.

    Default value: **0**

  - stat_state

    Determines whether table statistics are locked. If locked, the table statistics cannot be updated.

    Value range: **locked** and **unlock**

    Default value: **unlock**

  - statistic_granularity

    Records the default **partition_mode** when the table analyzes statistics. For details about **partition_mode**, see **ANALYZE|ANALYSE**. This parameter is invalid for non-partitioned tables.

    Value range: See the value range of **partition_mode**.

    Default value: **AUTO**

  - immediate_delete

    This parameter can be set only for vector database tables. After this parameter is set, index data is searched and deleted immediately when heap table data is deleted in the Astore scenario. If this parameter is set for an ordinary table, the Astore deletion and update performance deteriorates.

    Value range: **false** and **true**

    Default value: **false**

- **WITHOUT OIDS**

  It is equivalent to **WITH(OIDS=FALSE)**.

- **ON COMMIT { PRESERVE ROWS | DELETE ROWS }**

  ON COMMIT determines what to do when you commit a temporary table creation operation. Currently, **PRESERVE ROWS** and **DELETE ROWS** are supported.

  - **PRESERVE ROWS** (default): No special action is taken at the ends of transactions. The temporary table and its table data are unchanged.
  - **DELETE ROWS**: All rows in the temporary table will be deleted at the end of each transaction block.

- **COMPRESS | NOCOMPRESS**

  If you specify **COMPRESS** in the CREATE TABLE statement, the compression feature is triggered in case of a bulk insert operation. If this feature is

enabled, a scan is performed for all tuple data within the page to generate a dictionary and then the tuple data is compressed and stored. If **NOCOMPRESS** is specified, the table is not compressed. Row-store tables do not support compression.

Default value: **NOCOMPRESS**, that is, tuple data is not compressed before storage.

- **[ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ON (EXPR)]]**

  When creating a table, you can call ILM ADD POLICY ROW STORE COMPRESS ADVANCED ROW to add an advanced compression policy for row store. For example, CREATE TABLE t1 (a int) ILM ADD POLICY ROW STORE COMPRESS ADVANCED ROW AFTER 3 DAY OF NO MODIFICATION ON (a != 0) indicates that the **t1** table is created and the advanced compression policy is added: rows that are not modified in three days and **a != 0** is specified.

  - AFTER n { day | month | year } OF NO MODIFICATION: indicates the rows that are not modified in *n* days, months, or years.
  - ON (EXPR): indicates the row-level expression, which is used to determine whether a row is hot or cold.

- **TABLESPACE tablespace_name**

  Specifies the tablespace where the new table is created. If not specified, the default tablespace is used.

- **CONSTRAINT constraint_name**

  Specifies the name of a column or table constraint. The optional constraint clauses specify constraints that new or updated rows must satisfy for an INSERT or UPDATE operation to succeed.

  There are two ways to define constraints:

  - A column constraint is defined as part of a column definition, and it is bound to a particular column.
  - A table constraint is not bound to a particular column but can apply to more than one column.

- **NOT NULL**

  Forbids NULL values in columns.

- **NULL**

  Allows to contain NULL values. This is the default setting.

  This clause is only provided for compatibility with non-standard SQL databases. It is not recommended.

- **CHECK (expression)**

  Specifies an expression producing a Boolean result where the INSERT or UPDATE operation of rows can succeed only when the expression result is **TRUE** or **UNKNOWN**; otherwise, an error is thrown and the database is not altered.

  A CHECK constraint specified as a column constraint should reference only the column's value, while an expression in a table constraint can reference multiple columns.

  NOTE

  <>NULL and **!=NULL** are invalid in an expression. Change them to **IS NOT NULL**.

- **DEFAULT default_expr**

  Assigns a default data value to a column. The value can be any variable-free expressions. (Subqueries and cross-references to other columns in the current table are not allowed.) The data type of the default expression must match the data type of the column.

  The default expression will be used in any insert operation that does not specify a value for the column. If there is no default value for a column, then the default value is null.

- **ON UPDATE update_expr**

  The ON UPDATE clause is an attribute constraint of a column.

  When an UPDATE operation is performed on a tuple in a table, if new values of updated columns are different from old values in the table, column values with this attribute but not in updated columns are automatically updated to the current timestamp. If new values of updated columns are the same as old values in the table, column values with this attribute but not in updated columns remain unchanged. If columns with this attribute are in updated columns, column values are updated according to the specified update value.

  NOTE

  - This attribute can be specified only in B-compatible database 5.7 (that is, **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**).
  - In terms of syntax, **update_expr** supports three keywords: CURRENT_TIMESTAMP, LOCALTIMESTAMP, and NOW(). You can also specify or not specify the precision of a keyword with parentheses. For example, ON UPDATE CURRENT_TIMESTAMP(), ON UPDATE CURRENT_TIMESTAMP(5), ON UPDATE LOCALTIMESTAMP(), and ON UPDATE LOCALTIMESTAMP(6). If the keyword does not contain parentheses or contains empty parentheses, the precision is 0. The NOW keyword cannot contain parentheses. The three types of keywords are synonyms of each other and have the same attribute effect.
  - This attribute can be specified only for columns of the following types: timestamp, datetime, date, time without time zone, smalldatetime, and abstime.
  - The CREATE TABLE AS syntax does not inherit the column attributes.
  - The CREATE TABLE LIKE syntax can use INCLUDING UPDATE or EXCLUDING UPDATE to inherit or exclude a constraint. The LIKE syntax is inherited from the LIKE syntax of PostgreSQL. Currently, the ILM policy information of the old table cannot be copied.
  - The precision specified by this attribute can be different from the precision specified by the type in the corresponding column. After the column value is updated through this attribute, the minimum precision is displayed. For example, CREATE TABLE t1 (col1 timestamp(6) ON UPDATE CURRENT_TIMESTAMP(3));. If the UPDATE syntax triggers the attribute to take effect, three decimal places in the value of **col1** are displayed after the update.
  - The same column cannot be specified for this attribute and the generated column constraint at the same time.
  - This attribute cannot be specified for the partition key in a partitioned table.

- **GENERATED ALWAYS AS (generation_expr) [STORED]**

  This clause creates a column as a generated column. The value of the generated column is calculated by **generation_expr** when data is written (inserted or updated). **STORED** indicates that the value of the generated column is stored as a common column.

NOTE

- The STORED keyword can be omitted, which has the same semantics as not omitting STORED.
- The generation expression cannot refer to data other than the current row in any way. The generation expression cannot reference other generation columns or system columns. The generation expression cannot return a result set. No subquery, aggregate function, or window function can be used. The function called by the generation expression can only be an immutable function.
- Default values cannot be specified for generated columns.
- The generated column cannot be used as a part of the partition key.
- Do not specify the generated column and the CASCADE, SET NULL, and SET DEFAULT actions of the ON UPDATE constraint clause together. Do not specify the generated column and the SET NULL, and SET DEFAULT actions of the ON DELETE constraint clause together.
- The method of modifying and deleting generated columns is the same as that of common columns. Delete the common column that the generated column depends on. The generated column is automatically deleted. The type of the column on which the generated column depends cannot be changed.
- The generated column cannot be directly written. In the INSERT or UPDATE statement, values cannot be specified for generated columns, but the keyword DEFAULT can be specified.
- The permission control for generated columns is the same as that for common columns.

- **GENERATED [ALWAYS | BY DEFAULT [ON NULL]] AS IDENTITY [(identity_options)]**

  Creates a column as an IDENTITY column. An implicit sequence is automatically created based on identity_options and attached to a specified column. When data is inserted, the value obtained from the sequence is automatically assigned to the column.

  – **GENERATED [ALWAYS] AS IDENTITY**: Only identity values provided by the sequence generator can be inserted into this column. User-specified values cannot be inserted into this column.

  – **GENERATED BY DEFAULT AS IDENTITY**: User-specified values are preferentially inserted into this column. If no value is specified, the identity values provided by the sequence generator are inserted.

  – **GENERATED BY DEFAULT ON NULL AS IDENTITY**: User-specified values are preferentially inserted into this column. If a null value is specified or no value is specified, the identity values provided by the sequence generator are inserted.

  The optional identity_options clause can be used to override sequence options.

  – **increment**: specifies the implicit sequence step. If the value is a positive number, an ascending sequence is generated. If the value is a negative number, a descending sequence is generated. The default value is **1**.

  – **MINVALUE minvalue | NO MINVALUE | NOMINVALUE**: specifies the minimum value of an execution sequence. If **minvalue** is not specified or **NO MINVALUE** is specified, the default value of an ascending sequence is **1**, and the default value of a descending sequence is **$-10^{27} + 1$**. **NOMINVALUE** is equivalent to **NO MINVALUE**.

  – **MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE**: specifies the maximum value of an execution sequence. If **maxvalue** is not specified or

NO MAXVALUE is specified, the default value of an ascending sequence is $10^{28} - 1$, and the default value of a descending sequence is **–1**. **NOMAXVALUE** is equivalent to **NO MAXVALUE**.

- **start** specifies the start value of an implicit sequence. The default value for an ascending sequence is **minvalue** and that for a descending sequence is **maxvalue**.
- **cache**: specifies the number of sequences stored in the memory for quick access purposes. The default value is **1**, indicating that only one value can be generated at a time, that is, no cache is generated.
- **NOCACHE**: No value of the sequence is stored in advance.
- **CYCLE**: recycles sequences after the number of sequences reaches **maxvalue** or **minvalue**. If **NO CYCLE** is declared, any call to nextval returns an error after the sequence reaches its maximum or minimum value. **NOCYCLE** is equivalent to **NO CYCLE**. The default value is **NO CYCLE**.
- **SCALE**: enables sequence scalability. If specified, a numeric offset is appended to the beginning of the sequence to prevent duplicate items in the generated value. If **NOSCALE** is declared, sequence scalability is disabled. The default value is **NOSCALE**.
- **EXTEND**: extends the numeric offset length (default value: **6**), aligns the sequence generation value to **x** (default value: **6**) + **y** (maximum number of digits). **SCALE** must be specified when **EXTEND** is specified. If **NOEXTEND** is declared, the numeric offset length is not extended. The default value is **NOEXTEND**.

NOTE

- The IDENTITY column can only be of the smallint, integer, bigint, decimal, numeric, float, double precision, or real type.
- In A-compatible mode, when an IDENTITY column of the integer type is created, the IDENTITY column is of the numeric type by default.
- The method of changing an IDENTITY column is the same as that of changing a common column, but the type can be changed only to smallint, integer, bigint, decimal, numeric, float, double precision, or real.
- By default, the IDENTITY column has the NOT NULL constraint.
- A table can contain only one IDENTITY column.
- The method of deleting an IDENTITY column is the same as that of deleting a common column. When a column is deleted, the implicit sequence of the IDENTITY column is automatically deleted.
- The IDENTITY column cannot be specified together with the SET DEFAULT action.
- The type of the implicit sequence that is automatically created is LARGE SEQUENCE.
- You cannot use DROP LARGE SEQUENCE or ALTER LARGE SEQUENCE to modify the implicit sequence of an identity.
- After permission is granted to the table, data can be inserted properly. To modify the IDENTITY column, delete the IDENTITY attribute, or delete the IDENTITY column, you need to grant permission to the corresponding implicit sequence.
- The [SCALE [EXTEND | NOEXTED] | NOSCALE] clause is available only when an IDENTITY column is created in a centralized system in A-compatible mode.
- In a fully-encrypted database, the encrypted IDENTITY column cannot be specified during table creation.

- **AUTO_INCREMENT**

  Specifies an auto-increment column.

  If the value of this column is not specified (or the value of this column is set to **0**, **NULL**, or **DEFAULT**), the value of this column is automatically increased by the auto-increment counter.

  If this column is inserted or updated to a value greater than the current auto-increment counter, the auto-increment counter is updated to this value after the command is executed successfully.

  The initial auto-increment value is set by the AUTO_INCREMENT [=] value clause. If it is not set, the default value **1** is used.

  NOTE

  - The auto-increment column can be specified only when **sql_compatibility** is set to **'B'**.
  - The data type of the auto-increment column can only be integer, 4-byte or 8-byte floating point, or Boolean.
    - An error occurs if the auto-increment continues after an auto-increment value reaches the maximum value of a column data type.
  - Each table can have only one auto-increment column.
  - It is recommended that the auto-increment column be the first column of an index. Otherwise, an alarm is generated when a table is created, and errors may occur when some operations are performed on a table that contains auto-increment columns, for example, ALTER TABLE EXCHANGE PARTITION.
  - The DEFAULT value cannot be specified for an auto-increment column.
  - The expression of the CHECK constraint cannot contain auto-increment columns, and the expression for generating columns cannot contain auto-increment columns.
  - You can specify that the auto-increment column can be NULL. If it is not specified, the auto-increment column contains the NOT NULL constraint by default.
  - When a table containing an auto-increment column is created, a sequence that depends on the column is created as an auto-increment counter. You are not allowed to modify or delete the sequence using sequence-related functions. You can view the value of the sequence.
  - Sequences are not created for auto-increment columns in local temporary tables.
  - The auto-increment and refresh operations of the auto-increment counter are not rolled back.
    - Before data is inserted into a table, **0** or **NULL** triggers auto-increment. After data is inserted or updated to a table, the auto-increment counter is updated. If an error is reported after auto-increment and data is not inserted or updated to the table, the auto-increment counter does not roll back. Subsequent insert statements trigger auto-increment based on the auto-increment counter. As a result, the values of the auto-increment columns in the table are discontinuous.
    - If you insert or import reserved auto-increment cache values in batches, the values in the auto-increment column may be discontinuous. For details, see the description of the **auto_increment_cache** parameter.

- **[DEFAULT] CHARACTER SET | CHARSET [=] default_charset**

  Specifies the default character set of a table. If this parameter is specified separately, the default collation of the table is set to the default collation of the specified character set.

  This syntax is supported only when **sql_compatibility** is set to **'B'**.

- **[DEFAULT] COLLATE [=] default_collation**

  Specifies the default collation of a table. If this parameter is specified separately, the default character set of the table is set to the character set corresponding to the specified collation.

  This syntax is supported only when **sql_compatibility** is set to **'B'**. For details about the collation, see **Table 7-220**.

  NOTE

  If the character set or collation of a table is not explicitly specified and the default character set or collation of the schema is specified, the character set or collation of the table is inherited from the schema. If the default character set or collation of a schema does not exist, the character set and collation of the table inherit the character set and collation of the current database when **b_format_behavior_compat_options** contains **'default_collation'**.

- **UNIQUE [KEY] index_parameters**

  Specifies that a group of one or more columns of a table can contain only unique values.

  UNIQUE KEY can be used only when **sql_compatibility** is set to **'B'**, which has the same semantics as UNIQUE.

- **UNIQUE [index_name][USING method]({{ column_name [(length)] | (expression) } [ASC | DESC] }[, ...]) index_parameters**

  Specifies that a group of one or more columns of a table can contain only unique values.

  For the purpose of a unique constraint, null is not considered equal.

  column_name (length) indicates the prefix key. For details, see **column_name (length)**.

  index_name indicates the index name.

  NOTICE

  - The **index_name** is supported only in B-compatible databases (that is, **sql_compatibility** set to **'B'**).
  - For a unique key constraint, if both constraint_name and index_name are specified, index_name is used as the index name.

- **PRIMARY KEY index_parameters**

  PRIMARY KEY [USING method] ({ column_name [ASC | DESC] } [, ...]) index_parameters

  Specifies that a column or columns of a table can contain only unique (non-duplicate) and non-null values.

  Only one primary key can be specified for a table.

- **REFERENCES reftable [(refcolumn)] [MATCH matchtype] [ON DELETE action] [ON UPDATE action] (column constraint)**

  FOREIGN KEY (column_name [, ...]) REFERENCES reftable [(refcolumn [, ...])] [MATCH matchtype] [ON DELETE action] [ON UPDATE action] (table constraint)

  The foreign key constraint requires that the group consisting of one or more columns in the new table should contain and match only the referenced

column values in the referenced table. If **refcolumn** is omitted, the primary key of **reftable** is used. The referenced column should be the only column or primary key in the referenced table. A foreign key constraint cannot be defined between a temporary table and a permanent table.

There are three types of matching between a reference column and a referenced column:

- **MATCH FULL**: A column with multiple foreign keys cannot be **NULL** unless all foreign key columns are **NULL**.
- **MATCH SIMPLE** (default): Any unexpected foreign key column can be **NULL**.
- **MATCH PARTIAL**: This option is not supported currently.

In addition, when certain operations are performed on the data in the referenced table, the operations are performed on the corresponding columns in the new table. **ON DELETE**: specifies the operations to be executed after a referenced row in the referenced table is deleted. **ON UPDATE**: specifies the operation to be performed when the referenced column data in the referenced table is updated. Possible responses to the **ON DELETE** and **ON UPDATE** clauses are as follows:

- **NO ACTION** (default): When a foreign key is deleted or updated, an error indicating that the foreign key constraint is violated is created. If the constraint is deferrable and there are still any referenced rows, this error will occur when the constraint is checked.
- **RESTRICT**: When a foreign key is deleted or updated, an error indicating that the foreign key constraint is violated is created. It is the same as **NO ACTION** except that the constraint is not deferrable.
- **CASCADE**: deletes any row that references the deleted row from the new table, or update the column value of the referenced row in the new table to the new value of the referenced column.
- **SET NULL**: sets the referenced columns to **NULL**.
- **SET DEFAULT**: sets the referenced columns to their default values.

- **DEFERRABLE | NOT DEFERRABLE**

  Determines whether the constraint can be deferred. A constraint that is not deferrable will be checked immediately after every command. Checking of constraints that are deferrable can be postponed until the end of the transaction using the **SET CONSTRAINTS** command. **NOT DEFERRABLE** is the default value. Currently, only UNIQUE constraints, primary key constraints, and foreign key constraints accept this clause. All the other constraints are not deferrable.

- **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

  If a constraint is deferrable, this clause specifies the default time to check the constraint.

  - If the constraint is **INITIALLY IMMEDIATE** (default value), it is checked after each statement.
  - If the constraint is **INITIALLY DEFERRED**, it is checked only at the end of the transaction.

  The constraint check time can be altered using the **SET CONSTRAINTS** statement.

- **USING INDEX TABLESPACE tablespace_name**

  Allows selection of the tablespace in which the index associated with a **UNIQUE** or **PRIMARY KEY** constraint will be created. If not specified, the index is created in **default_tablespace**. If **default_tablespace** is empty, the default tablespace of the database is used.

- **ENCRYPTION_TYPE = encryption_type_value**

  For the encryption type in the ENCRYPTED WITH constraint, the value of **encryption_type_value** is **DETERMINISTIC** or **RANDOMIZED**.

Examples

- Temporary table

```
-- Create a temporary table.
gaussdb=# CREATE TEMP TABLE test_t1(
    id       CHAR(7),
    name     VARCHAR(20),
    province VARCHAR(60),                    -- Province
    country   VARCHAR(30) DEFAULT 'China'        -- Country
);

-- Insert data into the current session.
gaussdb=# INSERT INTO test_t1 VALUES ('0000009','Jack','Guangzhou','China');

-- The data in the temporary table is visible only in the current session. Therefore, the table does not contain data in another session.
gaussdb=# SELECT * FROM test_t1;
 id | name | age
----+------+-----
(0 rows)
-- Create a temporary table and specify that this table is deleted when the transaction is committed.
gaussdb=# CREATE TEMPORARY TABLE test_t2(
    id       CHAR(7),
    name     VARCHAR(20),
    province VARCHAR(60),                    -- Province
    country   VARCHAR(30) DEFAULT 'China'        -- Country
) ON COMMIT DELETE ROWS;
-- Delete the table.
gaussdb=# DROP TABLE test_t1;
gaussdb=# DROP TABLE test_t2;
```

- Specifying the character set and collation during table creation

```
-- Create table t1. Set the default character set of t1 to utf8mb4 and the default collation to utf8mb4_bin. Set the character set and collation of the c1 column to the default values of the table. Set the character set of the c2 column to utf8mb4, and its collation to utf8mb4_unicode_ci.
gaussdb=# CREATE TABLE t1(c1 text, c2 text charset utf8mb4 collate utf8mb4_unicode_ci) charset utf8mb4 collate utf8mb4_bin;
```

- IF NOT EXISTS keyword

  If this keyword is used, NOTICE is reported when the table does not exist. If this keyword is not used, ERROR is reported. In either case, the table fails to be created.

```
gaussdb=# CREATE TABLE test_t3(id INT);

-- Create a table named test_t3.
gaussdb=# CREATE TABLE test_t3(id INT);
ERROR:  relation "test_t3" already exists in schema "public"
DETAIL:  creating new table with existing name in the same schema

-- Use the IF NOT EXISTS keyword.
gaussdb=# CREATE TABLE IF NOT EXISTS test_t3(id INT);
NOTICE:  relation "test_t3" already exists, skipping
CREATE TABLE

-- Delete the table.
gaussdb=# DROP TABLE test_t3;
```


- Specifying a tablespace during table creation

```
-- Create a tablespace.
gaussdb=# CREATE TABLESPACE ds_tbs1 RELATIVE LOCATION 'tablespace/tablespace_1';

-- Specify a tablespace when creating a table.
gaussdb=# CREATE TABLE test(id CHAR(7), name VARCHAR(20)) TABLESPACE ds_tbs1;

-- Delete the table and tablespace.
gaussdb=# DROP TABLE test;
gaussdb=# DROP TABLESPACE ds_tbs1;
```

- Specifying the AUTO_INCREMENT column during table creation

```
-- Create a table and specify an auto-increment column. The column starts from 10.
gaussdb=# CREATE DATABASE test DBCOMPATIBILITY = 'B';
gaussdb=# \c test
test=# CREATE TABLE test_autoinc(col int primary key AUTO_INCREMENT, col1 int)
AUTO_INCREMENT = 10;

-- You are advised to use the auto-increment column as the first column of the index to create an
index.
test=# CREATE INDEX test_autoinc_ai ON test_autoinc(col);

-- Enter NULL to trigger auto-increment. The auto-increment value is 10.
test=# INSERT INTO test_autoinc(col, col1) VALUES(NULL,1);

-- Enter 100. The auto-increment is not triggered. After the insertion is successful, the auto-increment
count is updated to 100.
test=# INSERT INTO test_autoinc(col, col1) VALUES(100,2);

-- Enter 0 to trigger auto-increment. The auto-increment value is 101.
test=# INSERT INTO test_autoinc(col, col1) VALUES(0,3);

test=# SELECT col,col1 FROM test_autoinc ORDER BY 2,1;
 col | col1
-----+------
  10 |    1
 100 |    2
 101 |    3
(3 rows)

-- Delete.
test=# DROP TABLE test_autoinc;
-- Switch to the default database. Change the database name based on actual situation.
test=# \c postgres
gaussdb=# DROP DATABASE test;
```

- Creating a table using CREATE TABLE ... LIKE

```
-- Create the source table t1.
gaussdb=# CREATE TABLE t1(col INT);
CREATE TABLE

gaussdb=# \d t1
      Table "public.t1"
 Column |  Type   | Modifiers
--------+---------+-----------
 col    | integer |

-- Create the target table t2.
gaussdb=# CREATE TABLE t2(LIKE t1);
CREATE TABLE

gaussdb=# \d t2
      Table "public.t2"
 Column |  Type   | Modifiers
--------+---------+-----------
 col    | integer |

-- Delete.
gaussdb=# DROP TABLE t1,t2;
```


Examples of Creating a Table and Adding Constraints to the Table

- Non-null constraints

  If no value is specified for a column with a non-null constraint when data is added, an error is reported. You can add non-null constraints to multiple columns in a table.

```
-- Create a table and add a non-null constraint to the id column.
gaussdb=# CREATE TABLE test_t4(
    id       CHAR(7) NOT NULL,
    name     VARCHAR(20),
    province VARCHAR(60),                    -- Province
    country   VARCHAR(30) DEFAULT 'China'        -- Country
);

-- If the value of id is not specified or is NULL during data insertion, the non-null constraint is
triggered. As a result, the insertion fails.
gaussdb=# INSERT INTO test_t4 (name,province) VALUES ('scott','Shanghai');
ERROR:  null value in column "id" violates not-null constraint
DETAIL:  Failing row contains (null, scott, Shanghai, China)

-- Delete the table.
gaussdb=# DROP TABLE test_t4;
```

- Unique constraint

  The keyword UNIQUE is used to add a unique constraint to a column. When data is inserted, the constraint is triggered if the column is duplicate. Multiple **NULL** values are not duplicate values. When a unique constraint is added, a unique index is automatically added. You can add unique constraints to multiple columns in a table.

```
-- Create a table to add unique constraints.
gaussdb=# CREATE TABLE test_t5(
    id       CHAR(7) UNIQUE USING INDEX TABLESPACE pg_default, -- You can specify a tablespace or
use the default tablespace.
    name     VARCHAR(20),
    province VARCHAR(60),                    -- Province
    country   VARCHAR(30) DEFAULT 'China'        -- Country
);

-- You can also use the following method to manually name a unique constraint and add constraints
for multiple columns:
gaussdb=# CREATE TABLE test_t6(
    id       CHAR(7),
    name     VARCHAR(20),
    province VARCHAR(60),                     -- Province
    country   VARCHAR(30) DEFAULT 'China',        -- Country
    CONSTRAINT unq_test_id UNIQUE (id,name)
);
-- When data with duplicate IDs is inserted, constraints are triggered. As a result, the insertion fails.
gaussdb=# INSERT INTO test_t5(id) VALUES('0000010');
INSERT 0 1
gaussdb=# INSERT INTO test_t5(id) VALUES('0000010');
ERROR:  duplicate key value violates unique constraint "test_t5_id_key"
DETAIL:  Key (id)=(0000010) already exists.
-- The constraint is not triggered when data whose id is NULL is inserted for multiple times.
gaussdb=# INSERT INTO test_t5(id) VALUES (NULL);
INSERT 0 1
gaussdb=# INSERT INTO test_t5(id) VALUES (NULL);
INSERT 0 1

gaussdb=# SELECT * FROM test_t5;
   id    | name | province | country
---------+------+----------+---------
 0000010 |      |          | China
         |      |          | China
         |      |          | China
```



```
-- Delete the table.
gaussdb=# DROP TABLE test_t5;
gaussdb=# DROP TABLE test_t6;
```

- Primary key constraints

  The keyword PRIMARY KEY is used to add a unique constraint to a column. The column must be unique and cannot be empty. When a primary key constraint is added, a unique index is automatically created for the table, and a non-null constraint is automatically added for the column.

  Only one primary key constraint can be defined in each table.

```
-- Create a table and add a primary key constraint to the table.
gaussdb=# CREATE TABLE test_t6(
    id       CHAR(7) PRIMARY KEY,
    name     VARCHAR(20),
    province VARCHAR(60),                    -- Province
    country   VARCHAR(30) DEFAULT 'China'        -- Country
);
gaussdb=# INSERT INTO test_t6 (id,name,province) VALUES ('0000001','july','Beijing');

-- You can also use the following method to manually name a unique constraint and add constraints
for multiple columns:
gaussdb=# CREATE TABLE test_t7(
    id       CHAR(7),
    name     VARCHAR(20),
    province VARCHAR(60),                    -- Province
    country   VARCHAR(30) DEFAULT 'China',        -- Country
    CONSTRAINT pk_test_t6_id PRIMARY KEY (id,name)
);
-- Insert data whose id is NULL to trigger the constraint.
gaussdb=# INSERT INTO test_t6 (id,name,province) VALUES (NULL,'july','Beijing');
ERROR:  null value in column "id" violates not-null constraint
DETAIL:  Failing row contains (null, july, Beijing, China).

-- Insert data with duplicate id values to trigger the constraint.
gaussdb=# INSERT INTO test_t6 (id,name,province) VALUES ('0000001','ben','Shanghai');
ERROR:  duplicate key value violates unique constraint "test_t6_pkey"
DETAIL:  Key (id)=(0000001) already exists.
-- Delete the table.
gaussdb=# DROP TABLE test_t6;
gaussdb=# DROP TABLE test_t7;
```

- Check constraints

  The keyword CHECK adds a check constraint to a column. The check constraint must reference one or more columns in the table, and the result returned by the expression must be a Boolean value. Currently, expressions cannot contain subqueries. Both check and non-null constraints can be defined for the same column.

```
-- Create a table and add check constraints.
gaussdb=# CREATE TABLE test_t8 (
    id       CHAR(7),
    name     VARCHAR(20),
    age      INT CHECK(age > 0 AND age < 150)
);

-- Or run the following SQL statements to manually name check constraints and add check
constraints for one or more columns:
gaussdb=# CREATE TABLE test_t9 (
    id       CHAR(7),
    name     VARCHAR(20),
    age      INT,
    CONSTRAINT chek_test_t8_age CHECK(age > 0 AND age < 150)
);
-- If a value that does not comply with the expression is inserted, the check constraint is triggered. As
a result, the insertion fails.
gaussdb=# INSERT INTO test_t8 (id,name,age) VALUES ('0000007','scott',200);
```



```
ERROR:  new row for relation "test_t8" violates check constraint "test_t8_age_check"
DETAIL:  N/A
-- Delete the table.
gaussdb=# DROP TABLE test_t8;
gaussdb=# DROP TABLE test_t9;
```

- Foreign key constraints

  When two tables contain one or more public columns, you can use foreign key constraints to enforce the relationship between the two tables.

  - **FOREIGN KEY**: columns that are related to the referenced table.
  - **REFERENCES**: columns that are related to the referenced table and the original table.

  Foreign key constraints have the following features:

  - A column defined as a foreign key constraint can contain only the values of the referenced columns in other tables or **NULL**.
  - You can define foreign key constraints for one or more columns.
  - A column that defines a foreign key constraint and the corresponding referenced column can exist in the same table, which is called self-reference.
  - Both foreign keys and non-null constraints can be defined for the same column.
  - Columns to be applied in the main table must have primary key constraints or unique constraints.

```
-- Create a department table.
gaussdb=# CREATE TABLE dept(
    deptno INT PRIMARY KEY,
    loc VARCHAR(200)
);

-- Create an employee table and add foreign key constraints.
gaussdb=# CREATE TABLE emp(
    empno INT,
    name VARCHAR(50),
    deptno INT,
    CONSTRAINT fk_emp FOREIGN KEY (deptno) REFERENCES dept(deptno)
);
-- Insert data into the department table.
gaussdb=# INSERT INTO dept VALUES (10,'Beijing');
gaussdb=# INSERT INTO dept VALUES (20,'Beijing');
gaussdb=# INSERT INTO dept VALUES (30,'Shanghai');

-- Insert the deptno data that can be found in the department table into the employee table.
gaussdb=# INSERT INTO emp VALUES (1,'Bob',10);

-- Insert data whose deptno is NULL into the employee table.
gaussdb=# INSERT INTO emp VALUES (2,'Scott',NULL);

-- Insert the deptno data that cannot be found in the department table into the employee table.
gaussdb=# INSERT INTO emp VALUES (1,'Jack',999);
ERROR:  insert or update on table "emp" violates foreign key constraint "fk_emp"
DETAIL:  Key (deptno)=(999) is not present in table "dept".
-- View data.
gaussdb=# SELECT * FROM emp;
 empno | name  | deptno
-------+-------+--------
     1 | Bob   |     10
     2 | Scott |
(2 rows)
-- Delete the table.
gaussdb=# DROP TABLE emp;
gaussdb=# DROP TABLE dept;
```


Helpful Links

ALTER TABLE, **DROP TABLE**, and **CREATE TABLESPACE**

Suggestions

- UNLOGGED
  - The unlogged table and its indexes do not use the WAL log mechanism during data writing. Their write speed is much higher than that of ordinary tables. Therefore, they can be used for storing intermediate result sets of complex queries to improve query performance.
  - The unlogged table has no primary/standby mechanism. In case of system faults or abnormal breakpoints, data loss may occur. Therefore, the unlogged table cannot be used to store basic data.
- TEMPORARY | TEMP
  - A temporary table is automatically dropped at the end of a session.
- LIKE
  - The new table automatically inherits all column names, data types, and not-null constraints from this table. The new table is irrelevant to the original table after the creation.
- LIKE INCLUDING DEFAULTS
  - The default expressions are copied from the original table to the new table only if **INCLUDING DEFAULTS** is specified. The default behavior is to exclude default expressions, resulting in the copied columns in the new table having default values **NULL**.
- LIKE INCLUDING CONSTRAINTS
  - The **CHECK** constraints are copied from the original table to the new table only when **INCLUDING CONSTRAINTS** is specified. Other types of constraints are never copied to the new table. Not-null constraints are always copied to the new table. These rules also apply to column constraints and table constraints.
- LIKE INCLUDING INDEXES
  - Any indexes on the original table will not be created on the new table, unless the **INCLUDING INDEXES** clause is specified.
- LIKE INCLUDING STORAGE
  - **STORAGE** settings for the copied column definitions are copied only if **INCLUDING STORAGE** is specified. The default behavior is to exclude **STORAGE** settings.
- LIKE INCLUDING COMMENTS
  - If **INCLUDING COMMENTS** is specified, comments for the copied columns, constraints, and indexes are copied. The default behavior is to exclude comments.
- LIKE INCLUDING PARTITION
  - If **INCLUDING PARTITION** is specified, the partition definitions of the source table are copied to the new table, and the new table no longer uses the **PARTITION BY** clause. By default, the partition definition of the source table is not copied.

NOTICE

List and hash partitioned tables do not support **LIKE INCLUDING PARTITION**.

- LIKE INCLUDING RELOPTIONS
  - If **INCLUDING RELOPTIONS** is specified, the new table will copy the storage parameter (that is, **WITH** clause) of the source table. The default behavior is to exclude partition definition of the storage parameter of the original table.
- LIKE INCLUDING IDENTITY
  - If **INCLUDING IDENTITY** is specified, a SEQUENCE with the same **SEQUENCE** parameter as the source table is created to implement the identity, and the IDENTITY type is the same as that of the source table. By default, the identity of the source table is not copied.
- LIKE INCLUDING ALL
  - INCLUDING ALL contains the contents of INCLUDING DEFAULTS, INCLUDING CONSTRAINTS, INCLUDING INDEXES, INCLUDING STORAGE, INCLUDING COMMENTS, INCLUDING PARTITION, INCLUDING RELOPTIONS, and INCLUDING IDENTITY.
- ORIENTATION ROW
  - Creates a row-store table. Row-store applies to the OLTP service, which has many interactive transactions. An interaction involves many columns in the table. Using row-store can improve the efficiency.

7.12.8.46 CREATE TABLE AS

Description

Creates a table from the results of a query.

It creates a table and fills it with data obtained using SELECT. The table columns have the names and data types associated with the output columns of SELECT (except that you can override the SELECT output column names by giving an explicit list of new column names).

CREATE TABLE AS queries a source table once and writes the data in a new table. This table will not change according to the source table. In contrast, the view re-computes and defines its SELECT statement at each query.

Precautions

- This statement cannot be used to create a partitioned table.
- If an error occurs during table creation, after it is fixed, the system may fail to delete the disk files that are created before the last automatic clearance and whose size is not 0. This problem seldom occurs and does not affect system running of the database.
- If you add a row-level expression when adding or changing an ILM policy for a data object, note that the row-level expression supports only the functions listed in the whitelist. For details about the whitelist function list, see **Row Expression Function Whitelist**.

Syntax

```
CREATE [ [ GLOBAL | LOCAL ] [ TEMPORARY | TEMP ] | UNLOGGED ] TABLE [ IF NOT EXISTS ] table_name
    [ (column_name [, ...] ) ]
    [ { ENGINE [ = ] { InnoDB | 'InnoDB' | "InnoDB" } } [ [ , ] ... ] ]
    [ WITH ( {storage_parameter = value} [, ... ] ) ]
    [ ON COMMIT { PRESERVE ROWS | DELETE ROWS } ]
    [ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF
{ NO MODIFICATION } [ ON ( EXPR )]]
    [ TABLESPACE tablespace_name ]
    AS query
    [ WITH [ NO ] DATA ];
```

Parameters

- **UNLOGGED**

  Specifies that the table is created as an unlogged table. Data written to unlogged tables is not written to the WALs, which makes them considerably faster than ordinary tables. However, an unlogged table is automatically truncated after a crash or unclean shutdown. The contents of an unlogged table are also not replicated to standby servers. Any indexes created on an unlogged table are automatically unlogged as well.

  - Usage scenario: Unlogged tables do not ensure data security. Users can back up data before using unlogged tables; for example, users should back up the data before a system upgrade.
  - Troubleshooting: If data is missing in the indexes of unlogged tables due to some unexpected operations such as an unclean shutdown, users should rebuild the indexes with errors.

- **GLOBAL | LOCAL**

  When creating a temporary table, you can specify the **GLOBAL** or **LOCAL** keyword before **TEMP** or **TEMPORARY**. If the keyword **GLOBAL** is specified, GaussDB creates a global temporary table. Otherwise, GaussDB creates a local temporary table.

- **TEMPORARY | TEMP**

  If **TEMP** or **TEMPORARY** is specified, the created table is a temporary table. Temporary tables are classified into global temporary tables and local temporary tables. If the keyword **GLOBAL** is specified when a temporary table is created, the table is a global temporary table. Otherwise, the table is a local temporary table.

  The metadata of the global temporary table is visible to all sessions. After the sessions end, the metadata still exists. The user data, indexes, and statistics of a session are isolated from those of another session. Each session can only view and modify the data submitted by itself. Global temporary tables can be created using ON COMMIT PRESERVE ROWS or ON COMMIT DELETE ROWS. The former one creates a session-level table, where user data is automatically cleared when a session ends; the latter creates a transaction-level table, where user data is automatically cleared when a COMMIT or ROLLBACK operation is performed. If the **ON COMMIT** option is not specified during table creation, the session level is used by default. Different from local temporary tables, you can specify a schema that does not start with pg_temp when creating a global temporary table.

  A local temporary table is automatically dropped at the end of the current session. Therefore, you can create and use temporary tables in the current session as long as the connected database node in the session is normal.

Temporary tables are created only in the current session. If a DDL statement involves operations on temporary tables, a DDL error will be generated. Therefore, you are advised not to perform operations on temporary tables in DDL statements. TEMP is equivalent to TEMPORARY.

NOTICE

- Local temporary tables are visible to the current session through the schema starting with **pg_temp**. Users should not delete schema started with **pg_temp** or **pg_toast_temp**.
- If **TEMPORARY** or **TEMP** is not specified when you create a table and its schema is set to that starting with pg_temp in the current session, the table will be created as a temporary table.
- If global temporary tables or indexes are being used by other sessions, do not perform ALTER or DROP on the tables or indexes.
- The DDL of a global temporary table affects only the user data and indexes of the current session. For example, TRUNCATE, REINDEX, and ANALYZE are valid only for the current session.

- **IF NOT EXISTS**

  When IF NOT EXISTS is specified, the system checks whether a relationship with the same name already exists in the current schema before creating a table. It is not created and a NOTICE is returned if a relationship with the same name already exists. When IF NOT EXISTS is not specified and a relationship with the same name exists in the schema, an ERROR is returned.

- **table_name**

  Specifies the name of the table to be created.

  Value range: a string. It must comply with the **naming convention**.

- **column_name**

  Optional. Specifies the name of a column to be created in the new table. If no column name is specified, the columns in the new table are the same as those entered in the SELECT statement.

  Value range: a string. It must comply with the **naming convention**.

- **ENGINE**

  Supported in B-compatible mode and used only for syntax adaptation. Only InnoDB can be set and no actual effect is achieved.

- **WITH (storage_parameter [= value] [, ...])**

  Specifies an optional storage parameter for a table or an index. See details of parameters below.

  – FILLFACTOR

    The fill factor of a table is a percentage from 10 to 100. If the Ustore storage engine is used, the default value is **92**. If the Astore storage engine is used, the default value is **100** (completely filled). When a smaller fill factor is specified, INSERT operations fill table pages only to the indicated percentage. The remaining space on each page is reserved for updating rows on that page. This gives UPDATE a chance to place the updated copy of a row on the same page, which is more efficient than

placing it on a different page. For a table whose entries are never updated, setting the fill factor to **100** (complete filling) is the best choice, but in heavily updated tables a smaller fill factor would be appropriate. The parameter is only valid for row–store tables.

Value range: 10–100

– ORIENTATION

Value range:

ROW (default value): The data will be stored in rows.

– COMPRESSION

Specifies the compression level of table data. It determines the compression ratio and time. Generally, the higher the level of compression, the higher the ratio, the longer the time; and the lower the level of compression, the lower the ratio, the shorter the time. The actual compression ratio depends on the distribution mode of table data loaded.

Value range:

Row-store tables do not support compression.

- **ON COMMIT { PRESERVE ROWS | DELETE ROWS }**

  ON COMMIT determines what to do when you commit a temporary table creation operation. Currently, only **PRESERVE ROWS** and **DELETE ROWS** are supported.

  – **PRESERVE ROWS** (default): No special action is taken at the ends of transactions. The temporary table and its table data are unchanged.

  – **DELETE ROWS**: All rows in the temporary table will be deleted at the end of each transaction block.

- **[ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ON (EXPR)]]**

  When creating a table, you can call ILM ADD POLICY ROW STORE COMPRESS ADVANCED ROW to add an advanced compression policy for row store.

  – AFTER n { day | month | year } OF NO MODIFICATION: indicates the rows that are not modified in *n* days, months, or years.

  – ON (EXPR): indicates the row-level expression, which is used to determine whether a row is hot or cold.

- **TABLESPACE tablespace_name**

  Specifies that the new table will be created in the **tablespace_name** tablespace. If not specified, the default tablespace is used.

- **AS query**

  Specifies a **SELECT** or **VALUES** command, or an **EXECUTE** command that runs a prepared **SELECT** or **VALUES** query.

- **[WITH [NO] DATA]**

  Specifies whether the data produced by the query should be copied to the new table. By default, the data will be copied. If the value **NO** is used, only the table structure will be copied.

Examples

- If no column name is specified, the columns in the new table are the same as those queried in the SELECT statement.


```
-- Create the test1 table and insert two records into the table.
gaussdb=# CREATE TABLE test1(col1 int PRIMARY KEY,col2 varchar(10));
gaussdb=# INSERT INTO test1 VALUES (1,'col1'),(101,'col101');
-- Query the data whose col1 is less than 100 in the table.
gaussdb=# SELECT * FROM test1 WHERE col1 < 100;
 col1 | col2
------+------
    1 | col1
(1 row)

-- Create the test2 table and insert the queried data into the table.
gaussdb=# CREATE TABLE test2 AS SELECT * FROM test1 WHERE col1 < 100;

-- Query the structure of the test2 table.
gaussdb=# \d test2;
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 col1   | integer               |
 col2   | character varying(10) |
```

- Specify column names for the new table.

```
-- Use test1 to copy a new table test3 and specify column names.
gaussdb=# CREATE TABLE test3(c1,c2) AS SELECT * FROM test1;

-- Query the structure of the test3 table.
gaussdb=# \d test3
           Table "public.test3"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 c1     | integer               |
 c2     | character varying(10) |

-- Delete.
gaussdb=# DROP TABLE test1,test2,test3;
```

- Specify a compression policy for the new table.

```
gaussdb=# CREATE TABLE old_table (a int);

-- Enable the ILM feature of the database.
gaussdb=# ALTER DATABASE SET ILM = on;

gaussdb=# CREATE TABLE ilm_table
    ILM ADD POLICY ROW STORE COMPRESS ADVANCED
    ROW AFTER 3 MONTHS OF NO MODIFICATION
    AS (SELECT * FROM old_table);

-- Delete.
gaussdb=# DROP TABLE old_table,ilm_table;
```

Helpful Links

CREATE TABLE and **SELECT**

7.12.8.47 CREATE TABLE PARTITION

Description

Creates a partitioned table. Partitioning refers to splitting what is logically one large table into smaller physical pieces based on specific schemes. The table based on the logic is called a partitioned table, and each physical piece is called a partition. Data is stored on these physical partitions, instead of the logical partitioned table.

The common forms of partitioning include range partitioning, interval partitioning, hash partitioning, list partitioning, and value partitioning. Currently, row-store

tables support range partitioning, interval partitioning, hash partitioning, and list partitioning.

In range partitioning, a table is partitioned based on ranges defined by one or more columns, with no overlap between the ranges of values assigned to different partitions. Each range has a dedicated partition for data storage.

The partitioning policy for range partitioning refers to how data is inserted into partitions. Currently, range partitioning only allows the use of the range partitioning policy.

In range partitioning, a table is partitioned based on partition key values. If a record can be mapped to a partition, it is inserted into the partition; if it cannot, an error message is returned. Range partitioning is the most commonly used partitioning policy.

Interval partitioning is a special type of range partitioning. Compared with range partitioning, interval value definition is added. When no matching partition can be found for an inserted record, a partition can be automatically created based on the interval value.

Interval partitioning supports only table-based partitioning of a list where the data type can be TIMESTAMP[(p)] [WITHOUT TIME ZONE], TIMESTAMP[(p)] [WITH TIME ZONE] and DATE.

Interval partitioning policy: A record is mapped to a created partition based on the partition key value. If the record can be mapped to a created partition, the record is inserted into the corresponding partition. Otherwise, a partition is automatically created based on the partition key value and table definition information, and then the record is inserted into the new partition. The data range of the new partition is equal to the interval value.

In hash partitioning, a modulus and a remainder are specified for each partition based on a column in the table, and records to be inserted into the table are allocated to the corresponding partition, the rows in each partition must meet the following condition: The value of the partition key divided by the specified modulus generates the remainder specified for the partition key.

In hash partitioning, table is partitioned based on partition key values. If a record can be mapped to a partition, it is inserted into the partition; if it cannot, an error message is returned.

List partitioning is to allocate the records to be inserted into a table to the corresponding partition based on the key values in each partition. The key values do not overlap in different partitions. Create a partition for each group of key values to store corresponding data.

In list partitioning, table is partitioned based on partition key values. If a record can be mapped to a partition, it is inserted into the partition; if it cannot, an error message is returned.

Partitioning can provide several benefits:

- Query performance can be improved drastically in certain situations, particularly when most of the heavily accessed rows of the table are in a single partition or a small number of partitions. Partitioning narrows the range of data search and improves data access efficiency.

- In the case of an INSERT or UPDATE operation on most portions of a single partition, performance can be improved by taking advantage of continuous scan of that partition instead of partitions scattered across the whole table.
- Frequent loading or deletion operations on records in a separate partition can be accomplished by reading or deleting that partition. This not only improves performance but also avoids the VACUUM overload caused by bulk DELETE operations (hash partitions cannot be deleted).

Precautions

- If the constraint key of the unique constraint and primary key constraint contains all partition keys, a local index is created for the constraints. Otherwise, a global index is created.
- Currently, hash partitioning supports only single-column partition keys, and does not support multi-column partition keys.
- When you have the INSERT permission on an interval partitioned table, partitions can be automatically created when you run INSERT to write data to the table.
- In the PARTITION FOR (values) syntax for partitioned tables, values can only be constants.
- In the PARTITION FOR (values) syntax for partitioned tables, if data type conversion is required for values, you are advised to use forcible type conversion to prevent the implicit type conversion result from being inconsistent with the expected result.
- The maximum number of partitions is 1048575. Generally, it is impossible to create so many partitions, because too many partitions may cause insufficient memory. Create partitions based on the value of **local_syscache_threshold**. The memory used by the partitioned tables is about (number of partitions x 3/1024) MB. Theoretically, the memory occupied by the partitions cannot be greater than the value of **local_syscache_threshold**. In addition, some space must be reserved for other functions.
- Considering the impact on performance, it is recommended that the maximum number of partitions in a single table be less than or equal to 2000 and the number of subpartitions multiplied by (Number of local indexes + 1) be less than or equal to 10000.
- If the memory is insufficient due to too many partitions, the performance deteriorates sharply.
- Currently, the statement specifying a partition cannot perform global index scan.
- Data of the XML type cannot be used as partition keys or level-2 partition keys.
- If you add a row-level expression when adding or changing an ILM policy for a data object, note that the row-level expression supports only the functions listed in the whitelist. For details about the whitelist function list, see **Row Expression Function Whitelist**.

Syntax

```
CREATE TABLE [ IF NOT EXISTS ] partition_table_name
( [
    { column_name data_type [ CHARACTER SET | CHARSET charset ] [ COLLATE collation ]
```



```
[ column_constraint [ ... ] ]
   | table_constraint
   | LIKE source_table [ like_option [...] ] }[, ... ]
] )

   [ table_option [ [ , ] ... ] ]
   [ WITH ( {storage_parameter = value} [, ... ] ) ]
   [ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF
{ NO MODIFICATION } [ ON ( EXPR )]]
   [ TABLESPACE tablespace_name ]
    PARTITION BY {
      {RANGE [COLUMNS] (partition_key) [ INTERVAL (interval_expr) [ STORE IN (tablespace_name
[, ... ] ) ] ] [ PARTITIONS integer ] ( partition_less_than_item [, ... ] )} |
      {RANGE [COLUMNS] (partition_key) [ INTERVAL (interval_expr) [ STORE IN (tablespace_name
[, ... ] ) ] ] [ PARTITIONS integer ] ( partition_start_end_item [, ... ] )} |
      {LIST [COLUMNS] (partition_key) [ AUTOMATIC ] [ PARTITIONS integer ] ( PARTITION
partition_name VALUES [IN] (list_values) [ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED }
{ ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ ON ( EXPR )]] [TABLESPACE [=]
tablespace_name][, ... ])} |
      {{ HASH | KEY } (partition_key) [ PARTITIONS integer ] ( PARTITION partition_name [ ILM ADD
POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO
MODIFICATION } [ ON ( EXPR ) ]] [TABLESPACE [=] tablespace_name][, ... ])}
   } [ { ENABLE | DISABLE } ROW MOVEMENT ];
```

- **table_option** is as follows:

  ```
  { COMMENT [ = ] 'string' |
    AUTO_INCREMENT [ = ] value |
    [ DEFAULT ] CHARACTER SET | CHARSET [ = ] default_charset |
    [ DEFAULT ] COLLATE [ = ] default_collation }
  ```

- **column_constraint** is as follows:

  ```
  [ CONSTRAINT constraint_name ]
  { NOT NULL |
    NULL |
    CHECK ( expression ) |
    DEFAULT default_e xpr |
    ON UPDATE update_expr |
    GENERATED ALWAYS AS ( generation_expr ) [STORED] |
    GENERATED [ ALWAYS | BY DEFAULT [ ON NULL ] ] AS IDENTITY [ ( identity_options ) ] |
    AUTO_INCREMENT |
    COMMENT 'string' |
    UNIQUE [KEY] index_parameters |
    PRIMARY KEY index_parameters |
    REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
        [ ON DELETE action ] [ ON UPDATE action ] }
  [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
  ```

- **table_constraint** is as follows:

  ```
  [ CONSTRAINT [ constraint_name ] ]
  { CHECK ( expression ) |
    UNIQUE [ index_name ][ USING method ] ( { column_name [ ASC | DESC ] } [, ... ] )
  index_parameters |
    PRIMARY KEY [ USING method ] ( { column_name [ ASC | DESC ] } [, ... ] ) index_parameters |
    FOREIGN KEY [ index_name ] ( column_name [, ... ] ) REFERENCES reftable [ ( refcolumn [, ... ] ) ]
      [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ] [ ON UPDATE
  action ] }
  [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
  { [ COMMENT 'string' ] [ ... ] }
  ```

- **like_option** is as follows:

  ```
  { INCLUDING | EXCLUDING } { DEFAULTS | GENERATED | CONSTRAINTS | INDEXES | STORAGE |
  COMMENTS | RELOPTIONS| UPDATE | IDENTITY | ALL }
  ```

- **index_parameters** is as follows:

  ```
  [ WITH ( {storage_parameter = value} [, ... ] ) ]
  [ USING INDEX TABLESPACE tablespace_name ]
  ```

- partition_less_than_item:

  ```
  PARTITION partition_name VALUES LESS THAN {( { partition_value | MAXVALUE } [,...] ) |
  MAXVALUE } [ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day |
  month | year } OF { NO MODIFICATION } [ ON ( EXPR ) ]] [TABLESPACE [=] tablespace_name]
  ```


- partition_start_end_item:

```
PARTITION partition_name {
      {START(partition_value) END (partition_value) EVERY (interval_value)} |
      {START(partition_value) END ({partition_value | MAXVALUE})} |
      {START(partition_value)} |
      {END({partition_value | MAXVALUE})}
} [ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year }
OF { NO MODIFICATION } [ ON ( EXPR ) ]] [TABLESPACE [=] tablespace_name]
```

- update_expr:

```
{ CURRENT_TIMESTAMP | LOCALTIMESTAMP | NOW() }
```

Parameters

- **IF NOT EXISTS**

  Sends a notice instead of throwing an error, if a table with the same name exists.

- **partition_table_name**

  Specifies the name of a partitioned table.

  Value range: a string. It must comply with the **naming convention**.

- **column_name**

  Specifies the name of a column to be created in the new table.

  Value range: a string. It must comply with the **naming convention**.

- **data_type**

  Specifies the data type of the column.

- **COLLATE collation**

  Assigns a collation to the column (which must be of a collatable data type). If no collation is specified, the default collation is used. You can run the **SELECT * FROM pg_collation;** command to query collation rules from the pg_collation system catalog. The default collation rule is the row starting with **default** in the query result.

- **CONSTRAINT constraint_name**

  Specifies the name of a column or table constraint. The optional constraint clauses specify constraints that new or updated rows must satisfy for an INSERT or UPDATE operation to succeed.

  There are two ways to define constraints:

  – A column constraint is defined as part of a column definition, and it is bound to a particular column.

  – A table constraint is not bound to a particular column but can apply to more than one column. The constraint_name is optional in B-compatible mode (**sql_compatibility = 'B'**). For other modes, constraint_name must be added.

- **index_name**

  Index name

NOTICE

- The index_name is supported only in B-compatible databases (that is, **sql_compatibility = 'B'**).
- For foreign key constraints, if constraint_name and index_name are specified at the same time, constraint_name is used as the index name.
- For a unique key constraint, if both constraint_name and index_name are specified, index_name is used as the index name.

- **USING method**

  Specifies the name of the index method to be used.

  For details about the value range, see USING method in **Parameters**.

NOTICE

- The USING method is supported only in B-compatible databases (that is, **sql_compatibility = 'B'**).
- In B-compatible mode, if USING method is not specified, the default index method is btree for ASTORE or UB-tree for USTORE.

- **ASC | DESC**

  ASC specifies an ascending (default) sort order. **DESC** specifies a descending sort order.

NOTICE

The ASC|DESC is supported only in B-compatible databases (that is, **sql_compatibility = 'B'**).

- **LIKE source_table [like_option ...]**

  Specifies a table from which the new table automatically copies all column names, their data types, and their not-null constraints.

  The new table and original table are decoupled after creation is complete. Changes to the original table will not be applied to the new table, and it is not possible to include data of the new table in scans of the original table.

  - Default expressions for the copied column definitions will be copied only if **INCLUDING DEFAULTS** is specified. The default behavior is to exclude default expressions, resulting in the copied columns in the new table having default values **NULL**.
  - If **INCLUDING UPDATE** is specified, the **ON UPDATE CURRENT_TIMESTAMP** attribute of the original table column is copied to the new table column. By default, this attribute is not copied.
  - If **INCLUDING GENERATED** is specified, the generated expression of the original table column is copied to the new table. By default, the generated expression is not copied.
  - Not-null constraints are always copied to the new table. CHECK constraints will only be copied if INCLUDING CONSTRAINTS is specified;

other types of constraints will never be copied. These rules also apply to column constraints and table constraints.

The copied columns and constraints are not merged with similarly named columns and constraints. If the same name is specified explicitly or in another **LIKE** clause, an error is reported.

- Any indexes on the original table will not be created on the new table, unless the INCLUDING INDEXES clause is specified.
- If **INCLUDING STORAGE** is specified, the **STORAGE** setting of the original table column is also copied. By default, the **STORAGE** setting is not included.
- If **INCLUDING COMMENTS** is specified, comments of the original table columns, constraints, and indexes are also copied. The default behavior is to exclude comments.
- If **INCLUDING RELOPTIONS** is specified, the storage parameters (WITH clauses) of the original table are also copied to the new table. The default behavior is to exclude partition definition of the storage parameter of the original table.
- If INCLUDING IDENTITY is specified, the IDENTITY function of the original table is copied to the new table, and a SEQUENCE with the same SEQUENCE parameter as that of the original table is created. By default, the IDENTITY function of the original table is not copied.
- INCLUDING ALL contains the contents of INCLUDING DEFAULTS, INCLUDING UPDATE, INCLUDING CONSTRAINTS, INCLUDING INDEXES, INCLUDING STORAGE, INCLUDING COMMENTS, INCLUDING PARTITION, INCLUDING RELOPTIONS, and INCLUDING IDENTITY.

NOTICE

- The CREATE TABLE table_name LIKE source_table syntax is supported only when **sql_compatibility** is set to **'B'** (B-compatible database), **b_format_version** is set to **5.7**, and **b_format_dev_version** is set to **s2**.
- In a B-compatible database, if **b_format_version** is set to **5.7** and **b_format_dev_version** is set to **s2**, **INCLUDING** and **EXCLUDING** cannot be specified. In this case, it is equivalent to specify **INCLUDING ALL** by default.

- **AUTO_INCREMENT [=] value**

  This clause specifies an initial value for an auto-increment column. The value must be a positive integer and cannot exceed $2^{127} - 1$.

NOTICE

This clause takes effect only when **sql_compatibility** is set to **'B'**.

- **COMMENT [=] 'string'**
  - The COMMENT [=] 'string' clause is used to add comments to a table.
  - The COMMENT 'string' in column_constraint indicates that comments are added to a column.

  - The COMMENT 'string' in table_constraint indicates that comments are added to the indexes corresponding to the primary key and unique key.

  For details, see **COMMENT [=] 'string'**.

- **CHARACTER SET | CHARSET charset**

  Specifies the character set of a table column. If this parameter is specified separately, the collation of the table column is set to the default collation of the specified character set.

  This syntax is supported only in B-compatible databases (that is, **sql_compatibility = 'B'**).

- **COLLATE collation**

  Assigns a collation to the column (which must be of a collatable data type). If no collation is specified, the default collation is used. You can run the **SELECT * FROM pg_collation** command to query collation rules from the pg_collation system catalog. The default collation rule is the row starting with **default** in the query result. In B-compatible databases (that is, **sql_compatibility = 'B'**), **utf8mb4_bin**, **utf8mb4_general_ci**, **utf8mb4_unicode_ci**, **binary**, **gbk_chinese_ci**, **gbk_bin**, **gb18030_chinese_ci**, and **gb18030_bin** are also supported.

- **WITH (storage_parameter [= value] [, ...])**

  Specifies an optional storage parameter for a table or an index. Optional parameters are as follows:

  - FILLFACTOR

    The fill factor of a table is a percentage from 10 to 100. If the Ustore storage engine is used, the default value is **92**. If the Astore storage engine is used, the default value is **100** (completely filled). When a smaller fill factor is specified, INSERT operations fill table pages only to the indicated percentage. The remaining space on each page is reserved for updating rows on that page. This gives UPDATE a chance to place the updated copy of a row on the same page, which is more efficient than placing it on a different page. For a table whose entries are never updated, setting the fill factor to **100** (complete filling) is the best choice, but in heavily updated tables a smaller fill factor would be appropriate.

    Value range: 10–100

  - ORIENTATION

    Determines the data storage mode of the table.

    Value range:

    - **ROW** (default value): The data will be stored in rows.

      NOTICE

      orientation cannot be modified.

  - STORAGE_TYPE

    Specifies the storage engine type. This parameter cannot be modified once it is set.

    Value range:

- **USTORE** indicates that tables support the in-place update storage engine. Note that the **track_counts** and **track_activities** parameters must be enabled when the Ustore table is used. Otherwise, space bloating may occur.
- **ASTORE** indicates that tables support the append-only storage engine.

Default value:

If this parameter is not specified, the **enable_default_ustore_table** parameter determines the storage engine mode, which is inplace-update storage by default.

– COMPRESSION

- Row-store tables do not support compression.

– segment

The data is stored in segment-page mode. This parameter supports only row-store tables. Temporary tables and unlogged tables are not supported.

Value range: **on** and **off**

Default value: **off**

– statistic_granularity

Specifies the default value of **partition_mode** of a table during statistics analysis. For details about **partition_mode**, see **ANALYZE|ANALYSE**. This parameter is invalid for non-partitioned tables.

Value range: See the value range of **partition_mode**.

Default value: **AUTO**.

– enable_tde

Specifies that the table is an encrypted table. The database automatically encrypts the data in the encryption table before storing it. Before using this parameter, ensure that TDE function has been enabled using the GUC parameter **enable_tde** and the information for accessing the key service has been set using the GUC parameter **tde_key_info**. For details about how to use this parameter, see section "Transparent Data Encryption" in *Feature Guide*. This parameter applies only to row-store tables, segment-page tables, hash bucket tables, temporary tables, and unlogged tables.

Value range: **on** and **off** When **enable_tde** is set to **on**, **key_type**, **tde_cmk_id**, and **dek_cipher** are automatically generated by the database and cannot be manually specified or modified.

Default value: **off**

– encrypt_algo

Specifies the encryption algorithm of the encryption table. This parameter must be used together with **enable_tde**.

Value range: a string. The value can be **AES_128_CTR** or **SM4_CTR**.

Default value: null if **enable_tde** and **AES_128_CTR** if **enable_tde** is set.

– dek_cipher

Specifies the DEK ciphertext. After a user sets the **enable_tde** parameter for a table, the database automatically generates a data key.

Value range: a string

Default value: null

– key_type

Specifies the type of the master key. After the **enable_tde** parameter is set for a table, the database automatically obtains the master key type from the GUC parameter **tde_key_info**.

Value range: a string

Default value: null

– cmk_id

Specifies the ID of the master key. After the **enable_tde** parameter is set for a table, the database automatically obtains the master key ID from the GUC parameter **tde_key_info**.

Value range: a string

Default value: null

- **[ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ON (EXPR)]]**

  When creating a table, you can call ILM ADD POLICY ROW STORE COMPRESS ADVANCED ROW to add an advanced compression policy for row store. Partitions inherit the policy of the table.

  – AFTER n { day | month | year } OF NO MODIFICATION: indicates the rows that are not modified in *n* days, months, or years.

  – ON (EXPR): indicates the row-level expression, which is used to determine whether a row is hot or cold.

- **TABLESPACE tablespace_name**

  Specifies that the new table will be created in the **tablespace_name** tablespace. If not specified, the default tablespace is used.

- **PARTITION BY RANGE [COLUMNS] (partition_key)**

  Creates a range partition. **partition_key** is the name of the partition key.

  The COLUMNS keyword can be used only when **sql_compatibility** is set to **'B'**. The semantics of PARTITION BY RANGE COLUMNS is the same as that of PARTITION BY RANGE.

  (1) Assume that the VALUES LESS THAN syntax is used.

  NOTICE

  If the VALUES LESS THAN clause is used, a range partitioning policy supports a partition key with up to 16 columns.

  Data types supported by the partition keys are as follows: TINYINT, SMALLINT, INTEGER, BIGINT, DECIMAL, NUMERIC, REAL, DOUBLE PRECISION, CHARACTER VARYING(n), VARCHAR(n), CHARACTER(n), CHAR(n), CHARACTER, CHAR, TEXT, NVARCHAR, NVARCHAR2, NAME, TIMESTAMP[(p)] [WITHOUT TIME ZONE], TIMESTAMP[(p)] [WITH TIME ZONE], and DATE.

  (2) Assume that the START END syntax is used.

NOTICE

In this case, only one partition key is supported.

Data types supported by the partition key are as follows: TINYINT, SMALLINT, INTEGER, BIGINT, DECIMAL, NUMERIC, REAL, DOUBLE PRECISION, TIMESTAMP[(p)] [WITHOUT TIME ZONE], TIMESTAMP[(p)] [WITH TIME ZONE], and DATE.

(3) Assume that the INTERVAL syntax is used.

NOTICE

If the INTERVAL clause is specified, the range partitioning supports only one partition key.

In this case, the data types supported by the partition key are TIMESTAMP[(p)] [WITHOUT TIME ZONE], TIMESTAMP[(p)] [WITH TIME ZONE] and DATE.

- **PARTITION partition_name VALUES LESS THAN {({ partition_value | MAXVALUE } [,...]) | MAXVALUE }**

  Specifies the information of partitions. **partition_name** is the name of a range partition. **partition_value** is the upper limit of a range partition, and the value depends on the type of **partition_key**. *MAXVALUE* usually specifies the upper limit of the last range partition.

  NOTICE

  - Each partition requires an upper limit.
  - The data type of the upper limit must be the same as that of the partition key.
  - In a partition list, partitions are arranged in ascending order of upper limits. A partition with a smaller upper limit value is placed before another partition with a larger one.
  - MAVALUE that is not in parentheses can be used only when **sql_compatibility** is set to **'B'** and can have only one partition key.

- **PARTITION partition_name {START (partition_value) END (partition_value) EVERY (interval_value)} | {START (partition_value) END (partition_value|MAXVALUE)} | {START(partition_value)} | {END (partition_value | MAXVALUE)**}

  Specifies the information of partitions.

  - **partition_name**: name or name prefix of a range partition. It is the name prefix only in the following cases (assuming that **partition_name** is **p1**):

    - If **START**+**END**+**EVERY** is used, the names of partitions will be defined as **p1_1**, **p1_2**, and the like. For example, if **PARTITION p1 START(1) END(4) EVERY(1)** is defined, the generated partitions are

[1, 2), [2, 3), and [3, 4), and their names are **p1_1**, **p1_2**, and **p1_3**. In this case, **p1** is a name prefix.

  - If the defined statement is in the first place and has **START** specified, the range (*MINVALUE*, **START**) will be automatically used as the first actual partition, and its name will be **p1_0**. The other partitions are then named **p1_1**, **p1_2**, and the like. For example, if **PARTITION p1 START(1), PARTITION p2 START(2)** is defined, generated partitions are (*MINVALUE*, 1), [1, 2), and [2, *MAXVALUE*), and their names will be **p1_0**, **p1_1**, and **p2**. In this case, **p1** is a name prefix and **p2** is a partition name. **MINVALUE** means the minimum value.

- **partition_value**: start value or end value of a range partition. The value depends on **partition_key** and cannot be *MAXVALUE*.
- **interval_value**: width of each partition for dividing the [**START**, **END**) range. It cannot be *MAXVALUE*. If the value of (**END** – **START**) divided by **EVERY** has a remainder, the width of only the last partition is less than the value of **EVERY**.
- *MAXVALUE* usually specifies the upper limit of the last range partition.

NOTICE

1. If the defined statement is in the first place and has **START** specified, the range (*MINVALUE*, **START**) will be automatically used as the first actual partition.
2. The START END syntax must comply with the following rules:
   - The value of **START** (if any, same for the following situations) in each **partition_start_end_item** must be smaller than that of **END**.
   - In two adjacent **partition_start_end_item** statements, the value of the first **END** must be equal to that of the second **START**.
   - The value of **EVERY** in each **partition_start_end_item** must be a positive number (in ascending order) and must be smaller than **END** minus **START**.
   - Each partition includes the start value (unless it is *MINVALUE*) and excludes the end value. The format is as follows: [**START**, **END**).
   - Partitions created by the same **partition_start_end_item** belong to the same tablespace.
   - If **partition_name** is a name prefix of a partition, the length must not exceed 57 bytes. If there are more than 57 bytes, the prefix will be automatically truncated.
   - When creating or modifying a partitioned table, ensure that the total number of partitions in the table does not exceed the maximum value **1048575**.
3. In statements for creating partitioned tables, START END and LESS THAN cannot be used together.
4. The START END syntax in a partitioned table creation SQL statement will be replaced by the VALUES LESS THAN syntax when gs_dump is executed.

- **INTERVAL (interval_expr) [STORE IN (tablespace_name [, ...])]**

Defines interval partitioning.

- **interval_expr**: interval for automatically creating partitions. The value must comply with the value type of **partition_key**. Currently, only the numeric type and date/time type are supported, for example, **1 day** and **1 month**.
- STORE IN (tablespace_name [, ...]): Specifies the list of tablespaces for storing automatically created partitions. If this parameter is specified, the automatically created partitions are cyclically selected from the tablespace list. Otherwise, the default tablespace of the partitioned table is used.

- **PARTITION BY LIST [COLUMNS] (partition_key)**

  Create a list partition. **partition_key** is the name of the partition key.

  The keyword COLUMNS can be used only when **sql_compatibility** is set to **'B'**. The semantics of PARTITION BY LIST COLUMNS is the same as that of PARTITION BY LIST.

  - A list partitioning policy supports a partition key with up to 16 columns.
  - For the clause syntax VALUES [IN] (list_values), if **list_values** contains the key values of the corresponding partition, it is recommended that the number of key values of each partition be less than or equal to 64.
  - The clause VALUES IN can be used only when **sql_compatibility** is set to **'B'**. The semantics is the same as that of **VALUES**.

  Partition keys support the following data types: TINYINT, SMALLINT, INTEGER, BIGINT, NUMERIC, VARCHAR(n), CHAR, BPCHAR, NVARCHAR, NVARCHAR2, TIMESTAMP[(p)] [WITHOUT TIME ZONE], TIMESTAMP[(p)] [WITH TIME ZONE], and DATE. The number of partitions cannot exceed 1048575.

- **PARTITION BY HASH(partition_key)**

  Create a hash partition. **partition_key** is the name of the partition key.

  For **partition_key**, the hash partitioning policy supports only one column of partition keys.

  Partition keys support the following data types: TINYINT, SMALLINT, INTEGER, BIGINT, NUMERIC, VARCHAR(n), CHAR, BPCHAR, TEXT, NVARCHAR, NVARCHAR2, TIMESTAMP[(p)] [WITHOUT TIME ZONE], TIMESTAMP[(p)] [WITH TIME ZONE], and DATE. The number of partitions cannot exceed 1048575.

- **PARTITION BY KEY(partition_key)**

  This parameter can be used only when **sql_compatibility** is set to **'B'**. The semantics is the same as that of **PARTITION BY HASH(partition_key)**.

- **AUTOMATIC**

  If keyword **AUTOMATIC** is specified when a table is created, the automatic partitioning function of list partitions is enabled. By default, the automatic partitioning function is disabled. Only list partitions can use the automatic partitioning function.

  After the automatic partitioning function is enabled, if the inserted data cannot match an existing partition, an independent partition is automatically created.

- **PARTITIONS integer**

  Specifies the number of partitions.

integer indicates the number of partitions. The value must be an integer greater than 0 and cannot be greater than 1048575.

- When this clause is specified after the range and list partitions, each partition must be explicitly defined, and the number of defined partitions must be equal to the integer value. This clause can be specified after the range and list partitions only when **sql_compatibility** is set to **'B'**.
- When this clause is specified after the hash and key partitions, if the definition of each partition is not listed, an *integer* number of partitions are automatically generated. The automatically generated partition name is "p+number", and the number ranges from 0 to *integer* minus 1. The tablespace of the partition is the tablespace of the table by default. If each partition definition is explicitly defined, the number of defined partitions must be the same as the value of *integer*. If neither the partition definition nor the number of partitions is specified, a unique partition is created.

- **{ ENABLE | DISABLE } ROW MOVEMENT**

  Sets row movement.

  If the tuple value is updated on the partition key during the UPDATE operation, the partition where the tuple is located is altered. Setting this parameter enables error messages to be reported or movement of the tuple between partitions.

  Value range:

  - **ENABLE** (default value): Row movement is enabled.
  - **DISABLE**: Row movement is disabled.

  If the row movement is enabled, an error may be reported when UPDATE and DELETE operations are performed concurrently. The causes are as follows:

  The old data is marked as deleted in the UPDATE and DELETE operations. If the row movement is enabled, the cross-partition update occurs when the partition key is updated, the kernel marks the old data in the old partition as deleted and adds a data to the new partition. As a result, the new data cannot be found by querying the old data.

  If data in the same row is concurrently operated, the cross-partition and non-cross-partition data results have different behaviors in the following three concurrency scenarios: UPDATE and UPDATE concurrency, DELETE and DELETE concurrency, as well as UPDATE and DELETE concurrency.

  a. For non-cross-partition data, no error is reported for the second operation after the first operation is performed.

     - If the first operation is UPDATE, the latest data can be found and operated after the second operation is performed.
     - If the first operation is DELETE, the second operation is terminated if the current data is deleted and the latest data cannot be found.

  b. For the cross-partition data result, an error is reported for the second operation after the first operation is performed.

     - If the first operation is UPDATE, the second operation cannot find the latest data because the new data is in the new partition. Therefore, the second operation fails and an error is reported.

    - If the first operation is DELETE, performing the second operation can find that the current data is deleted and the latest data cannot be found, but cannot determine whether the operation of deleting the old data is UPDATE or DELETE. If the operation is UPDATE, an error is reported. If the operation is DELETE, the operation is terminated. To ensure the data correctness, an error is reported.

  If the UPDATE and UPDATE concurrency, and UPDATE and DELETE concurrency are performed, the error can be solved only when the operations are performed serially. If the DELETE and DELETE concurrency are performed, the error can be solved by disabling the row movement.

- **NOT NULL**

  Forbids NULL values in columns. **ENABLE** can be omitted.

- **NULL**

  Allows to contain NULL values. This is the default setting.

  This clause is only provided for compatibility with non-standard SQL databases. It is not recommended.

- **CHECK (condition) [NO INHERIT]**

  Specifies an expression producing a Boolean result where the INSERT or UPDATE operation of new or updated rows can succeed only when the expression result is **TRUE** or **UNKNOWN**; otherwise, an error is thrown and the database is not altered.

  A check constraint specified as a column constraint should reference only the column's values, while an expression in a table constraint can reference multiple columns.

  A constraint marked with **NO INHERIT** will not propagate to child tables.

  ENABLE can be omitted.

- **DEFAULT default_expr**

  Assigns a default data value to a column. The value can be any variable-free expressions. (Subqueries and cross-references to other columns in the current table are not allowed.) The data type of the default expression must match the data type of the column.

  The default expression will be used in any insert operation that does not specify a value for the column. If there is no default value for a column, then the default value is **NULL**.

- **ON UPDATE update_expr**

  The ON UPDATE clause is an attribute constraint of a column.

  When an UPDATE operation is performed on a tuple in a table, if new values of updated columns are different from old values in the table, column values with this attribute but not in updated columns are automatically updated to the current timestamp. If new values of updated columns are the same as old values in the table, column values with this attribute but not in updated columns remain unchanged. If columns with this attribute are in updated columns, column values are updated according to the specified update value.

NOTE

- This attribute can be specified only in B-compatible database 5.7 (that is, **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**).
- In terms of syntax, **update_expr** supports three keywords: CURRENT_TIMESTAMP, LOCALTIMESTAMP, and NOW(). You can also specify or not specify the precision of a keyword with parentheses. For example, ON UPDATE CURRENT_TIMESTAMP(), ON UPDATE CURRENT_TIMESTAMP(5), ON UPDATE LOCALTIMESTAMP(), and ON UPDATE LOCALTIMESTAMP(6). If the keyword does not contain parentheses or contains empty parentheses, the precision is 0. The NOW keyword cannot contain parentheses. The three types of keywords are synonyms of each other and have the same attribute effect.
- This attribute can be specified only for columns of the following types: timestamp, datetime, date, time without time zone, smalldatetime, and abstime.
- The CREATE TABLE AS syntax does not inherit the column attributes.
- The CREATE TABLE LIKE syntax can use INCLUDING UPDATE or EXCLUDING UPDATE to inherit or exclude a constraint. The LIKE syntax is inherited from the LIKE syntax of PostgreSQL. Currently, the ILM policy information of the old table cannot be copied.
- The precision specified by this attribute can be different from the precision specified by the type in the corresponding column. After the column value is updated through this attribute, the minimum precision is displayed. For example, CREATE TABLE t1 (col1 timestamp(6) ON UPDATE CURRENT_TIMESTAMP(3));. If the UPDATE syntax triggers the attribute to take effect, three decimal places in the value of **col1** are displayed after the update.
- The same column cannot be specified for this attribute and the generated column constraint at the same time.
- This attribute cannot be specified for the partition key in a partitioned table.

- **GENERATED ALWAYS AS (generation_expr) [STORED]**

  This clause creates a column as a generated column. The value of the generated column is calculated by **generation_expr** when data is written (inserted or updated). **STORED** indicates that the value of the generated column is stored as a common column.

NOTE

- The STORED keyword can be omitted, which has the same semantics as not omitting STORED.
- The generation expression cannot refer to data other than the current row in any way. The generation expression cannot reference other generation columns or system columns. The generation expression cannot return a result set. No subquery, aggregate function, or window function can be used. The function called by the generation expression can only be an immutable function.
- Default values cannot be specified for generated columns.
- The generated column cannot be used as a part of the partition key.
- Do not specify the generated column and the CASCADE, SET NULL, and SET DEFAULT actions of the ON UPDATE constraint clause together. Do not specify the generated column and the SET NULL, and SET DEFAULT actions of the ON DELETE constraint clause together.
- The method of modifying and deleting generated columns is the same as that of common columns. Delete the common column that the generated column depends on. The generated column is automatically deleted. The type of the column on which the generated column depends cannot be changed.
- The generated column cannot be directly written. In the INSERT or UPDATE statement, values cannot be specified for generated columns, but the keyword DEFAULT can be specified.
- The permission control for generated columns is the same as that for common columns.

- **GENERATED [ALWAYS | BY DEFAULT [ON NULL]] AS IDENTITY [(identity_options)]**

  Creates a column as an IDENTITY column. An implicit sequence is automatically created based on identity_options and attached to a specified column. When data is inserted, the value obtained from the sequence is automatically assigned to the column.

  – **GENERATED ALWAYS AS IDENTITY**: Only identity values provided by the sequence generator can be inserted into this column. User-specified values cannot be inserted into this column.

  – **GENERATED BY DEFAULT AS IDENTITY**: User-specified values are preferentially inserted into this column. If no value is specified, the identity values provided by the sequence generator are inserted.

  – **GENERATED BY DEFAULT ON NULL AS IDENTITY**: User-specified values are preferentially inserted into this column. If a null value is specified or no value is specified, the identity values provided by the sequence generator are inserted.

  The optional identity_options clause can be used to override sequence options.

  – **increment**: specifies the implicit sequence step. If the value is a positive number, an ascending sequence is generated. If the value is a negative number, a descending sequence is generated. The default value is **1**.

  – **MINVALUE minvalue | NO MINVALUE | NOMINVALUE**: specifies the minimum value of an execution sequence. If **minvalue** is not specified or **NO MINVALUE** is specified, the default value of an ascending sequence is **1**, and the default value of a descending sequence is **$-10^{27} + 1$**. **NOMINVALUE** is equivalent to **NO MINVALUE**.

  – **MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE**: specifies the maximum value of an execution sequence. If **maxvalue** is not specified or

NO MAXVALUE is specified, the default value of an ascending sequence is $10^{28} - 1$, and the default value of a descending sequence is **–1**. **NOMAXVALUE** is equivalent to **NO MAXVALUE**.

- **start** specifies the start value of an implicit sequence. The default value for an ascending sequence is **minvalue** and that for a descending sequence is **maxvalue**.
- **cache**: specifies the number of sequences stored in the memory for quick access purposes. The default value is **1**, indicating that only one value can be generated at a time, that is, no cache is generated.
- **NOCACHE**: No value of the sequence is stored in advance.
- **CYCLE**: recycles sequences after the number of sequences reaches **maxvalue** or **minvalue**. If **NO CYCLE** is declared, any call to nextval returns an error after the sequence reaches its maximum or minimum value. **NOCYCLE** is equivalent to **NO CYCLE**. The default value is **NO CYCLE**.
- **SCALE**: enables sequence scalability. If specified, a numeric offset is appended to the beginning of the sequence to prevent duplicate items in the generated value. If **NOSCALE** is declared, sequence scalability is disabled. The default value is **NOSCALE**.
- **EXTEND**: extends the numeric offset length (default value: **6**), aligns the sequence generation value to **x** (default value: **6**) + **y** (maximum number of digits). **SCALE** must be specified when **EXTEND** is specified. If **NOEXTEND** is declared, the numeric offset length is not extended. The default value is **NOEXTEND**.

NOTE

- The IDENTITY column can only be of the smallint, integer, bigint, decimal, numeric, float, double precision, or real type.
- In A-compatible mode, when an IDENTITY column of the integer type is created, the IDENTITY column is of the numeric type by default.
- The method of changing an IDENTITY column is the same as that of changing a common column, but the type can be changed only to smallint, integer, bigint, decimal, numeric, float, double precision, or real.
- By default, the IDENTITY column has the NOT NULL constraint.
- A table can contain only one IDENTITY column.
- The method of deleting an IDENTITY column is the same as that of deleting a common column. When a column is deleted, the implicit sequence of the IDENTITY column is automatically deleted.
- The IDENTITY column cannot be specified together with the SET DEFAULT action.
- The type of the implicit sequence that is automatically created is LARGE SEQUENCE.
- You cannot use DROP LARGE SEQUENCE or ALTER LARGE SEQUENCE to modify the implicit sequence of an identity.
- After permission is granted to the table, data can be inserted properly. To modify the IDENTITY column, delete the IDENTITY attribute, or delete the IDENTITY column, you need to grant permission to the corresponding implicit sequence.
- The [SCALE [EXTEND | NOEXTED] | NOSCALE] clause is available only when an IDENTITY column is created in a centralized system in A-compatible mode.
- In a fully-encrypted database, the encrypted IDENTITY column cannot be specified during table creation.

- **AUTO_INCREMENT**

  Specifies an auto-increment column.

  For details, see **AUTO_INCREMENT**.

- **UNIQUE [KEY] index_parameters**

  UNIQUE (column_name [, ...]) index_parameters

  Specifies that a group of one or more columns of a table can contain only unique values.

  For the purpose of a unique constraint, null is not considered equal.

  UNIQUE KEY can be used only when **sql_compatibility** is set to **'B'**, which has the same semantics as UNIQUE.

- **PRIMARY KEY index_parameters**

  PRIMARY KEY (column_name [, ...]) index_parameters

  Specifies that a column or columns of a table can contain only unique (non-duplicate) and non-null values.

  Only one primary key can be specified for a table.

- **DEFERRABLE | NOT DEFERRABLE**

  Determines whether the constraint can be deferred. A constraint that is not deferrable will be checked immediately after every command. Checking of constraints that are deferrable can be postponed until the end of the transaction using the **SET CONSTRAINTS** command. **NOT DEFERRABLE** is the default value. Currently, only UNIQUE constraints, primary key constraints, and foreign key constraints accept this clause. All the other constraints are not deferrable.

- **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

  If a constraint is deferrable, this clause specifies the default time to check the constraint.

  – If the constraint is **INITIALLY IMMEDIATE** (default value), it is checked after each statement.

  – If the constraint is **INITIALLY DEFERRED**, it is checked only at the end of the transaction.

  The constraint check time can be altered using the **SET CONSTRAINTS** command.

- **USING INDEX TABLESPACE tablespace_name**

  Allows selection of the tablespace in which the index associated with a **UNIQUE** or **PRIMARY KEY** constraint will be created. If not specified, the index is created in **default_tablespace**. If **default_tablespace** is empty, the default tablespace of the database is used.

Examples of Range Partitioning

- VALUES LESS THAN

```
-- Create a tablespace.
CREATE TABLESPACE tbs_test_range1_p1 RELATIVE LOCATION 'tbs_test_range1/tablespace_1';
CREATE TABLESPACE tbs_test_range1_p2 RELATIVE LOCATION 'tbs_test_range1/tablespace_2';
CREATE TABLESPACE tbs_test_range1_p3 RELATIVE LOCATION 'tbs_test_range1/tablespace_3';
CREATE TABLESPACE tbs_test_range1_p4 RELATIVE LOCATION 'tbs_test_range1/tablespace_4';

-- Create a partitioned table test_range1.
CREATE TABLE test_range1(
```



```
    id INT,
    info VARCHAR(20)
) PARTITION BY RANGE (id) (
    PARTITION p1 VALUES LESS THAN (200) TABLESPACE tbs_test_range1_p1,
    PARTITION p2 VALUES LESS THAN (400) TABLESPACE tbs_test_range1_p2,
    PARTITION p3 VALUES LESS THAN (600) TABLESPACE tbs_test_range1_p3,
    PARTITION pmax VALUES LESS THAN (MAXVALUE) TABLESPACE tbs_test_range1_p4
);

-- Insert 1000 data records.
INSERT INTO test_range1 VALUES(GENERATE_SERIES(1,1000),'abcd');

-- Check that the number of rows in the p1 partition is 199, that is, the record range is [1, 200).
SELECT COUNT(*) FROM test_range1 PARTITION (p1);
 count
-------
   199
(1 row)

-- Check that the number of rows in the p2 partition is 200, that is, the record range is [200, 400).
SELECT COUNT(*) FROM test_range1 PARTITION (p2);
 count
-------
   200
(1 row)

-- View the partition information.
SELECT a.relname, a.boundaries, b.spcname
FROM pg_partition a, pg_tablespace b
WHERE a.reltablespace = b.oid AND a.parentid = 'test_range1'::regclass;
 relname | boundaries |      spcname
---------+------------+--------------------
 p1      | {200}      | tbs_test_range1_p1
 p2      | {400}      | tbs_test_range1_p2
 p3      | {600}      | tbs_test_range1_p3
 pmax    | {NULL}     | tbs_test_range1_p4
(4 rows)

-- Delete.
DROP TABLE test_range1;
DROP TABLESPACE tbs_test_range1_p1;
DROP TABLESPACE tbs_test_range1_p2;
DROP TABLESPACE tbs_test_range1_p3;
DROP TABLESPACE tbs_test_range1_p4;
```

- START END

```
-- Create a partitioned table.
CREATE TABLE test_range2(
    id INT,
    info VARCHAR(20)
) PARTITION BY RANGE (id) (
    PARTITION p1 START(1) END(600) EVERY(200),
    PARTITION p2 START(600) END(800),
    PARTITION pmax START(800) END(MAXVALUE)
);

-- View the partition information.
SELECT relname, boundaries FROM pg_partition WHERE parentid = 'test_range2'::regclass AND
parttype = 'p' ORDER BY 1;
 relname | boundaries
---------+------------
 p1_0    | {1}
 p1_1    | {201}
 p1_2    | {401}
 p1_3    | {600}
 p2      | {800}
 pmax    | {NULL}
(6 rows)
```



```
-- Delete.
DROP TABLE test_range2;
```

Example of List Partitions

```
-- Create a list partitioned table.
CREATE TABLE test_list ( NAME VARCHAR ( 50 ), area VARCHAR ( 50 ) )
PARTITION BY LIST (area) (
    PARTITION p1 VALUES ('Beijing'),
    PARTITION p2 VALUES ('Shanghai'),
    PARTITION p3 VALUES ('Guangzhou'),
    PARTITION p4 VALUES ('Shenzhen'),
    PARTITION pdefault VALUES (DEFAULT)
);

-- Insert data.
INSERT INTO test_list VALUES ('bob', 'Shanghai'),('scott', 'Sichuan');

-- Query partition data.
SELECT * FROM test_list PARTITION (p2);
 name |   area
------+----------
 bob  | Shanghai
(1 row)
SELECT * FROM test_list PARTITION (pdefault);
 name  |  area
-------+---------
 scott | Sichuan
(1 row)

-- Delete.
DROP TABLE test_list;
```

Example of Hash Partitioning

```
-- Create a hash partitioned table and specify the number of partitions.
CREATE TABLE test_hash1(c1 int) PARTITION BY HASH(c1) PARTITIONS 3;

-- Create a hash partitioned table and specify the names of partitions.
CREATE TABLE test_hash2(c1 int) PARTITION BY HASH(C1)(
    PARTITION pa,
    PARTITION pb,
    PARTITION pc
);

-- View the partition information.
SELECT b.relname AS table_name,
       a.relname AS partition_name
FROM pg_partition a,
     pg_class b
WHERE b.relname LIKE 'test_hash%'
  AND a.parttype = 'p'
  AND a.parentid = b.oid;
 table_name | partition_name
------------+----------------
 test_hash1 | p2
 test_hash1 | p1
 test_hash1 | p0
 test_hash2 | pc
 test_hash2 | pb
 test_hash2 | pa
(6 rows)

-- Delete.
DROP TABLE test_hash1,test_hash2;
```


Helpful Links

ALTER TABLE PARTITION and **DROP TABLE**

7.12.8.48 CREATE TABLESPACE

Description

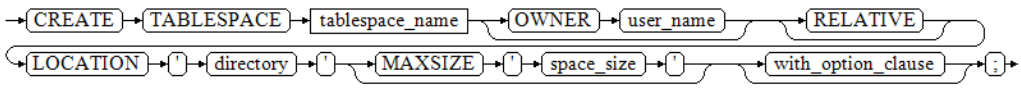
Creates a tablespace in a database.

Precautions

- The system administrator or a user who inherits the gs_role_tablespace permission of the built-in role can create a tablespace.
- Do not run **CREATE TABLESPACE** in a transaction block.
- If running **CREATE TABLESPACE** fails but the internal directory (or file) has been created, the directory (or file) will remain. You need to manually clear it before creating the tablespace again. If there are residual files of soft links for the tablespace in the data directory, delete the residual files, and then perform OM operations.
- **CREATE TABLESPACE** cannot be used for two-phase transactions. If it fails on some nodes, the execution cannot be rolled back.
- You are advised not to use user-defined tablespaces in the Huawei Cloud scenario. This is because user-defined tablespaces are usually used with storage media other than the main storage (storage device where the default tablespace is located, such as a disk) to isolate I/O resources that can be used by different services. Storage devices use standard configurations and do not have other available storage media in the Huawei Cloud scenario. If the user-defined tablespace is not properly used, the system cannot run stably for a long time and the overall performance is affected. Therefore, you are advised to use the default tablespace.

Syntax

```
CREATE TABLESPACE tablespace_name
    [ OWNER user_name ] [ RELATIVE ] LOCATION 'directory' [ MAXSIZE 'space_size' ]
    [with_option_clause];
```



with_option_clause for creating a general tablespace is as follows:

```
WITH ( {filesystem= { ' general ' | " general " | general } | address = { ' ip:port [, ... ] ' | " ip:port [, ... ] " } |
cfgpath = { ' path ' | " path " } | storepath = { ' rootpath ' | " rootpath " } | random_page_cost = { ' value ' | "
value " | value } | seq_page_cost = { ' value ' | " value " | value }}[, ... ])
```

Parameters

- **tablespace_name**

  Specifies name of a tablespace to be created.

  The tablespace name must be distinct from the name of any existing tablespace in the database and cannot start with "pg", which are reserved for system catalog spaces.

Value range: a string. It must comply with the **naming convention**.

- **OWNER user_name**

  Specifies the name of the user who will own the tablespace. If omitted, the default owner is the current user.

  Only system administrators can create tablespaces, but they can use the OWNER clause to assign ownership of tablespaces to non-system-administrators.

  Value range: a string. It must be an existing user.

- **RELATIVE**

  If this parameter is specified, a relative path is used. The location directory is relative to the data directory on each database node.

  Directory hierarchy: the relative path of the directory **/pg_location/** A relative path contains a maximum of two levels.

  If this parameter is not specified, the absolute tablespace path is used. The location directory must be an absolute path.

- **LOCATION directory**

  Specifies the directory for the table space. When creating an absolute tablespace path, ensure that the directory meets the following requirements:

  – The system user must have the read and write permissions on the directory, and the directory must be empty. If the directory does not exist, the system automatically creates it.
  – The directory must be an absolute path, and does not contain special characters, such as dollar sign ($) and greater-than sign (>).
  – The directory cannot be specified under the database data directory.
  – The directory must be a local path.

  Value range: a string. It must be a valid directory.

- **MAXSIZE 'space_size'**

  Specifies the maximum value of the tablespace in a single database node.

  Value range: a string consisting of a positive integer and unit. The unit can be KB, MB, GB, TB, or PB currently. The unit of parsed value is KB and cannot exceed the range that can be expressed in 8 bits, which is 1 KB to 9007199254740991 KB.

- **filesystem**

  Specifies the file system used by the tablespace.

  Value range:

  general: general file system.

  hdfs: Hadoop distributed file system. The current version does not support this function.

  Default value: **general**

- **random_page_cost**

  Specifies the time and resources required for randomly reading pages.

  Value range: 0 to 1.79769e+308

  Default value: value of the GUC parameter **random_page_cost**

- **seq_page_cost**

Specifies the time and resources required for reading pages in sequence.

Value range: 0 to 1.79769e+308

Default value: value of GUC parameter **seq_page_cost**

Examples

```
-- Create a tablespace.
gaussdb=# CREATE TABLESPACE tbs_location1 RELATIVE LOCATION 'test_tablespace/test_tablespace_1';

-- Create a tablespace and specify the maximum value.
gaussdb=# CREATE TABLESPACE tbs_location2 RELATIVE LOCATION 'test_tablespace/test_tablespace_2'
MAXSIZE '10G';

-- Query tablespace information.
gaussdb=# SELECT * FROM pg_tablespace WHERE spcname = 'tbs_location2';
    spcname    | spcowner | spcacl | spcoptions | spcmaxsize | relative
---------------+----------+--------+------------+------------+----------
 tbs_location2 |       10 |        |            | 10485760 K | t
(1 row)

-- Create a user joe.
gaussdb=# CREATE ROLE joe IDENTIFIED BY '********';

-- Create a tablespace and set its owner to user joe.
gaussdb=# CREATE TABLESPACE tbs_location3 OWNER joe RELATIVE LOCATION 'test_tablespace/
test_tablespace_3';

-- Delete tablespaces and users.
gaussdb=# DROP TABLESPACE tbs_location1;
gaussdb=# DROP TABLESPACE tbs_location2;
gaussdb=# DROP TABLESPACE tbs_location3;
gaussdb=# DROP ROLE joe;
```

Helpful Links

CREATE DATABASE, **CREATE TABLE**, **CREATE INDEX**, **DROP TABLESPACE**, and **ALTER TABLESPACE**

Suggestions

- create tablespace

  You are advised not to create tablespaces in a transaction.

7.12.8.49 CREATE TABLE SUBPARTITION

Description

Creates a level-2 partitioned table. Partitioning refers to splitting what is logically one large table into smaller physical pieces based on specific schemes. The table based on the logic is called a partitioned table, and each physical piece is called a partition. Data is stored on these physical partitions, instead of the logical partitioned table. For a level-2 partitioned table, the top-level node table and level-1 partitioned table are logical tables and do not store data. Only the level-2 partitioned (leaf node) stores data.

The partitioning solution of a level-2 partitioned table is a combination of the partitioning solutions of two level-1 partitions. For details about the partitioning solution of a level-1 partitioned table, see **CREATE TABLE PARTITION**.

Common combination solutions for level-2 partitioned tables include range-range partitioning, range-list partitioning, range-hash partitioning, list-range partitioning, list-list partitioning, list-hash partitioning, hash-range partitioning, hash-list partitioning, and hash-hash partitioning. Currently, level-2 partitioned tables can only be row-store tables.

Precautions

- A level-2 partitioned table has two partition keys, and each partition key supports only one column.
- If the constraint key of the unique constraint and primary key constraint contains all partition keys, a local index is created for the constraints. Otherwise, a global index is created. If a local unique index is created, all partition keys must be included.
- When a level-2 partitioned table is created, if the specified level-2 partition is not displayed under the level-1 partition, a level-2 partition with the same range is automatically created.
- The maximum total number of partitions (including level-1 and level-2 partitions) in a level-2 partitioned table is 1,048,575. Generally, you are advised not to create so many partitions. If the memory is insufficient due to too many partitions, the performance deteriorates sharply. Create partitions based on the value of **local_syscache_threshold**. The memory allocated to the level-2 partitioned tables can be calculated as follows: total number of partitions x 3/1024, in MB. Theoretically, the memory occupied by the partitions cannot be greater than the value of **local_syscache_threshold**. In addition, some space must be reserved for other functions.
- Considering the impact on performance, it is recommended that the maximum number of partitions in a single table be less than or equal to 2000 and the number of level-2 partitions multiplied by (Number of local indexes + 1) be less than or equal to 10000.
- Level-2 partitioned tables support only row store and do not support hash bucket.
- Clusters are not supported.
- When specifying a partition for query, for example, running **SELECT * FROM tablename PARTITION/SUBPARTITION** (*partitionname*), ensure that the keyword PARTITION and SUBPARTITION is correct. If they are incorrect, no error is reported during the query. In this case, the query is performed based on the table alias.
- Encrypted databases, ledger databases, and row-level security are not supported.
- In the PARTITION/SUBPARTITION FOR (VALUES) syntax for level-2 partitioned tables, VALUES can only be constants.
- In the PARTITION/SUBPARTITION FOR (VALUES) syntax for level-2 partitioned tables, if data type conversion is required for VALUES, you are advised to use forcible type conversion to prevent the implicit type conversion result from being inconsistent with the expected result.
- Currently, the statement specifying a partition cannot perform global index scan.
- If you add a row-level expression when adding or changing an ILM policy for a data object, note that the row-level expression supports only the functions

listed in the whitelist. For details about the whitelist function list, see **Row Expression Function Whitelist**.

Syntax

```
CREATE TABLE [ IF NOT EXISTS ] subpartition_table_name
(
{ column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
| table_constraint
| LIKE source_table [ like_option [...] ] }[, ... ]
)
[ table_option [ [ , ] ... ] ]
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ TABLESPACE tablespace_name ]
PARTITION BY {RANGE [ COLUMNS ] | LIST [ COLUMNS ] | HASH | KEY} (partition_key)[ PARTITIONS
integer ]
SUBPARTITION BY {RANGE | LIST | HASH | KEY} (subpartition_key) [ AUTOMATIC ] [ SUBPARTITIONS
integer ]
(
  PARTITION partition_name1 [ VALUES LESS THAN {(val1) | MAXVALUE} | VALUES [IN] (val1[, …]) ]
[ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF
{ NO MODIFICATION } [ ON ( EXPR ) ] ]
 [ TABLESPACE [=] tablespace ]
  [(
      { SUBPARTITION subpartition_name1 [ VALUES LESS THAN (val1_1) | VALUES (val1_1[, …])]
[ ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF
{ NO MODIFICATION } [ ON ( EXPR ) ] ]
 [ TABLESPACE [=] tablespace ] } [, ...]
  )][, ...]
)[ { ENABLE | DISABLE } ROW MOVEMENT ];
```

- **table_option** is as follows:

  ```
  { COMMENT [ = ] 'string' |
    AUTO_INCREMENT [ = ] value }
  ```

- Column constraint:

  ```
  [ CONSTRAINT constraint_name ]
  { NOT NULL |
    NULL |
    CHECK ( expression ) |
    DEFAULT default_e xpr |
    ON UPDATE update_expr |
    GENERATED ALWAYS AS ( generation_expr ) [STORED] |
    GENERATED [ ALWAYS | BY DEFAULT [ ON NULL ] ] AS IDENTITY [ ( identity_options ) ] |
    AUTO_INCREMENT |
    COMMENT 'string' |
    UNIQUE [KEY] index_parameters |
    PRIMARY KEY index_parameters |
    REFERENCES reftable [ ( refcolumn ) ] [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
        [ ON DELETE action ] [ ON UPDATE action ] }
  [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
  ```

- Table constraint:

  ```
  [ CONSTRAINT [ constraint_name ] ]
  { CHECK ( expression ) |
    UNIQUE  [ index_name ][ USING method ] ( { column_name [ ASC | DESC ] } [, ... ] )
  index_parameters |
    PRIMARY KEY  [ USING method ] ( { column_name [ ASC | DESC ] } [, ... ] ) index_parameters |
    FOREIGN KEY [ index_name ] ( column_name [, ... ] ) REFERENCES reftable [ ( refcolumn [, ... ] ) ]
      [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE action ] [ ON UPDATE
  action ] }
  [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
  { [ COMMENT 'string' ] [ ... ] }
  ```

- LIKE options:

  ```
  { INCLUDING | EXCLUDING } { DEFAULTS | GENERATED | CONSTRAINTS | INDEXES | STORAGE |
  COMMENTS | RELOPTIONS| UPDATE | IDENTITY | ALL }
  ```

- Index parameters:


```
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ USING INDEX TABLESPACE tablespace_name ]
```

- **update_expr** is as follows:

```
{ CURRENT_TIMESTAMP | LOCALTIMESTAMP | NOW() }
```

Parameters

- **IF NOT EXISTS**

  Does not throw an error if a relationship with the same name existed. A notice is issued in this case.

- **subpartition_table_name**

  Specifies the name of a level-2 partitioned table.

  Value range: a string. It must comply with the **naming convention**.

- **column_name**

  Specifies the name of a column to be created in the new table.

  Value range: a string. It must comply with the **naming convention**.

- **data_type**

  Specifies the data type of the column.

- **COLLATE collation**

  Assigns a collation to the column (which must be of a collatable data type). If no collation is specified, the default collation is used. You can run the **SELECT * FROM pg_collation;** command to query collation rules from the pg_collation system catalog. The default collation rule is the row starting with **default** in the query result.

- **CONSTRAINT constraint_name**

  Specifies the name of a column or table constraint. The optional constraint clauses specify constraints that new or updated rows must satisfy for an INSERT or UPDATE operation to succeed.

  There are two ways to define constraints:

  – A column constraint is defined as part of a column definition, and it is bound to a particular column.

  – A table constraint applies to multiple columns.

  NOTICE

  The constraint_name is optional in B-compatible mode (**sql_compatibility = 'B'**). For other modes, constraint_name must be added.

- **index_name**

  Index name

NOTICE

- The **index_name** is supported only in B-compatible databases (that is, **sql_compatibility** set to **'B'**).
- For foreign key constraints, if constraint_name and index_name are specified at the same time, constraint_name is used as the index name.
- For a unique key constraint, if both constraint_name and index_name are specified, index_name is used as the index name.

- **USING method**

  Specifies the name of the index method to be used.

  For details about the value range, see USING method in **Parameters**.

NOTICE

- The USING method is supported only in B-compatible databases (that is, **sql_compatibility** set to **'B'**).
- In B-compatible mode, if USING method is not specified, the default index method is btree for ASTORE or UB-tree for USTORE.

- **ASC | DESC**

  ASC specifies an ascending (default) sort order. **DESC** specifies a descending sort order.

NOTICE

The ASC|DESC is supported only in B-compatible databases (that is, **sql_compatibility** set to **'B'**).

- **LIKE source_table [like_option ...]**

  Specifies a table from which the new table automatically copies all column names, their data types, and their not-null constraints.

  The new table and original table are decoupled after creation is complete. Changes to the original table will not be applied to the new table, and it is not possible to include data of the new table in scans of the original table.

  - Default expressions for the copied column definitions will be copied only if **INCLUDING DEFAULTS** is specified. The default behavior is to exclude default expressions, resulting in the copied columns in the new table having default values null.
  - If **INCLUDING UPDATE** is specified, the **ON UPDATE CURRENT_TIMESTAMP** attribute of the original table column is copied to the new table column. By default, the generated expression is not copied.
  - If **INCLUDING GENERATED** is specified, the generated expression of the original table column is copied to the new table. By default, the generated expression is not copied.
  - Non-null constraints are always copied to the new table. CHECK constraints will only be copied if INCLUDING CONSTRAINTS is specified;

other types of constraints will never be copied. These rules also apply to column constraints and table constraints.

The copied columns and constraints are not merged with similarly named columns and constraints. If the same name is specified explicitly or in another **LIKE** clause, an error is reported.

- Any indexes on the original table will not be created on the new table, unless the INCLUDING INDEXES clause is specified.
- If **INCLUDING STORAGE** is specified, the **STORAGE** setting of the original table column is also copied. By default, the **STORAGE** setting is not included.
- If **INCLUDING COMMENTS** is specified, comments of the original table columns, constraints, and indexes are also copied. The default behavior is to exclude comments.
- If **INCLUDING RELOPTIONS** is specified, the storage parameters (WITH clauses) of the original table are also copied to the new table. The default behavior is to exclude partition definition of the storage parameter of the original table.
- If INCLUDING IDENTITY is specified, the IDENTITY function of the original table is copied to the new table, and a SEQUENCE with the same SEQUENCE parameter as that of the original table is created. By default, the IDENTITY function of the original table is not copied.
- INCLUDING ALL contains the contents of INCLUDING DEFAULTS, INCLUDING UPDATE, INCLUDING CONSTRAINTS, INCLUDING INDEXES, INCLUDING STORAGE, INCLUDING COMMENTS, INCLUDING PARTITION, INCLUDING RELOPTIONS, and INCLUDING IDENTITY.

- **AUTO_INCREMENT [=] value**

  This clause specifies an initial value for an auto-increment column. The value must be a positive integer and cannot exceed $2^{127} - 1$.

  NOTICE

  This clause takes effect only when **sql_compatibility** is set to **'B'**.

- **COMMENT [=] 'string'**
  - The COMMENT [=] 'string' clause is used to add comments to a table.
  - The COMMENT 'string' in column_constraint indicates that comments are added to a column.
  - The COMMENT 'string' in table_constraint indicates that comments are added to the indexes corresponding to the primary key and unique key.

  For details, see **COMMENT [=] 'string'**.

- **WITH (storage_parameter [= value] [, ...])**

  Specifies an optional storage parameter for a table or an index. Optional parameters are as follows:

  - FILLFACTOR

    The fill factor of a table is a percentage from 10 to 100. If the Ustore storage engine is used, the default value is **92**. If the Astore storage engine is used, the default value is **100** (completely filled). When a

smaller fill factor is specified, INSERT operations fill table pages only to the indicated percentage. The remaining space on each page is reserved for updating rows on that page. This gives UPDATE a chance to place the updated copy of a row on the same page, which is more efficient than placing it on a different page. For a table whose entries are never updated, setting the fill factor to **100** (complete filling) is the best choice, but in heavily updated tables a smaller fill factor would be appropriate.

Value range: 10–100

- ORIENTATION

  Determines the data storage mode of the table.

  Value range:

  - **ROW** (default value): The data will be stored in rows.

    NOTICE

    ORIENTATION cannot be modified.

- STORAGE_TYPE

  Specifies the storage engine type. This parameter cannot be modified once it is set.

  Value range:

  - **USTORE** indicates that tables support the in-place update storage engine.

    NOTICE

    To use Ustore tables, you need to enable the **track_counts** and **track_activities** parameters. Otherwise, space expansion occurs.

  - **ASTORE** indicates that tables support the append-only storage engine.

  Default value:

  If no table is specified, data is stored in inplace-update mode by default.

- COMPRESSION

  - Row-store tables do not support compression.

- segment

  The data is stored in segment-page mode. This parameter supports only row-store tables. Temporary tables and unlogged tables are not supported.

  Value range: **on** and **off**

  Default value: **off**

- statistic_granularity

Specifies the default partition_mode of a table during statistics analysis. For details about partition_mode, see **ANALYZE|ANALYSE**. This parameter is invalid for non-partitioned tables.

Value range: See the value range of **partition_mode**.

Default value: **AUTO**.

- enable_tde

  Specifies that the table is an encrypted table. The database automatically encrypts the data in the encrypted table before storing it. Before using this parameter, ensure that TDE function has been enabled using the GUC parameter **enable_tde** and the information for accessing the key service has been set using the GUC parameter **tde_key_info**. For details about how to use this parameter, see section "Transparent Data Encryption" in *Feature Guide*. This parameter applies only to row-store tables, segment-page tables, hash bucket tables, temporary tables, and unlogged tables.

  Value range: **on** and **off** When **enable_tde** is set to **on**, **key_type**, **tde_cmk_id**, and **dek_cipher** are automatically generated by the database and cannot be manually specified or modified.

  Default value: **off**

- encrypt_algo

  Specifies the encryption algorithm of the encryption table. This parameter must be used together with **enable_tde**.

  Value range: a string. The value can be **AES_128_CTR** or **SM4_CTR**.

  Default value: null if **enable_tde** and **AES_128_CTR** if **enable_tde** is set.

- dek_cipher

  Specifies the DEK ciphertext. After a user sets the **enable_tde** parameter for a table, the database automatically generates a data key.

  Value range: a string

  Default value: null

- key_type

  Specifies the type of the master key. After the **enable_tde** parameter is set for a table, the database automatically obtains the master key type from the GUC parameter **tde_key_info**.

  Value range: a string

  Default value: null

- cmk_id

  Specifies the ID of the master key. After the **enable_tde** parameter is set for a table, the database automatically obtains the master key ID from the GUC parameter **tde_key_info**.

  Value range: a string

  Default value: null

- **[ILM ADD POLICY ROW STORE { COMPRESS ADVANCED } { ROW } AFTER n { day | month | year } OF { NO MODIFICATION } [ON (EXPR)]]**

  When creating a table, you can call ILM ADD POLICY ROW STORE COMPRESS ADVANCED ROW to add an advanced compression policy for row store. Subpartitions inherit the policy of partitions.

  - AFTER n { day | month | year } OF NO MODIFICATION: indicates the rows that are not modified in *n* days, months, or years.
  - ON (EXPR): indicates the row-level expression, which is used to determine whether a row is hot or cold.

- **TABLESPACE tablespace_name**

  Specifies that the new table will be created in the **tablespace_name** tablespace. If not specified, the default tablespace is used.

- **PARTITION BY {RANGE [COLUMNS] | LIST [COLUMNS] | HASH | KEY} (partition_key)**
  - For **partition_key**, the partitioning policy supports only one column of partition keys.
  - The data types supported by the partition key are the same as those supported by the level-1 partitioned table.
  - The COLUMNS keyword can be used only when **sql_compatibility** is set to **'B'** and can be added only after RANGE or LIST. The semantics of RANGE COLUMNS is the same as that of RANGE, and the semantics of LIST COLUMNS is the same as that of LIST.
  - The KEY keyword can be used only when **sql_compatibility** is set to **'B'**. The meaning of KEY is the same as that of HASH.
- **SUBPARTITION BY {RANGE | LIST | HASH | KEY} (subpartition_key)**
  - For **subpartition_key**, the partitioning policy supports only one column of partition keys.
  - The data types supported by the partition key are the same as those supported by the level-1 partitioned table.
  - The KEY keyword can be used only when **sql_compatibility** is set to **'B'**. The meaning of KEY is the same as that of HASH.
- **AUTOMATIC**

  If keyword **AUTOMATIC** is specified when a table is created, the automatic partitioning function of list partitions is enabled. By default, the automatic partitioning function is disabled. Only list partitions can use the automatic partitioning function.

  After the automatic partitioning function is enabled, if the inserted data cannot match an existing partition, an independent partition is automatically created.

  NOTE

  Based on whether the automatic partitioning function is enabled for level-1 and level-2 partitions, the following situations may occur during data insertion:

  - The automatic expansion function is enabled for the level-1 partition, and any partition policy is used for the level-2 partition. If the inserted data does not match the level-1 partition, the corresponding level-1 partition is automatically created, and the corresponding level-2 partition is a full set.
  - The automatic partitioning function is enabled for the level-2 partition, and the level-1 partition is any partition policy. If the inserted data does not match the level-1 partition, the insertion fails. If the inserted data matches the level-1 partition but does not match the level-2 partition, the corresponding level-2 partition is automatically created.
  - The automatic partitioning function is enabled for both level-1 and level-2 partitions. If the inserted data does not match the level-1 or level-2 partition, the system automatically creates a partition at the corresponding level.

- **PARTITIONS integer**

  Specifies the number of partitions.

  integer indicates the number of partitions. The value must be an integer greater than 0 and cannot be greater than 1048575.

  – When this clause is specified after the range and list partitions, each partition must be explicitly defined, and the number of defined partitions must be equal to the integer value. This clause can be specified after the range and list partitions only when **sql_compatibility** is set to **'B'**.

  – When this clause is specified after the hash and key partitions, if the definition of each partition is not listed, an *integer* number of partitions are automatically generated. The automatically generated partition name is "p+number", and the number ranges from 0 to *integer* minus 1. The tablespace of the partition is the tablespace of the table by default. If each partition definition is explicitly defined, the number of defined partitions must be the same as the value of *integer*. If neither the partition definition nor the number of partitions is specified, a unique partition is created.

- **SUBPARTITIONS integer**

  Specifies the number of level-2 partitions.

  integer indicates the number of level-2 partitions. The value must be an integer greater than 0 and cannot be greater than 1048575.

  – This clause can be specified only for level-2 HASH and KEY partitions.

    ▪ If level-2 partitions are not defined, an *integer* number of level-2 partitions are automatically generated in each level-1 partition. The automatically generated level-2 partitions are named in the format of "level-1 partition name+sp+number", where the number ranges from 0 to *integer* minus 1. The tablespace of the partition is the tablespace of the table by default.

    ▪ If level-2 partitions are defined, the number of level-2 partitions must be the same as the value of **integer**.

    ▪ If level-2 partitions are not defined and the number of level-2 partitions is not specified, a unique level-2 partition is created.

- **{ ENABLE | DISABLE } ROW MOVEMENT**

  Specifies whether to enable row movement.

  If the tuple value is updated on the partition key during the UPDATE operation, the partition where the tuple is located is altered. Setting this parameter enables error messages to be reported or movement of the tuple between partitions.

  Value range:

  – **ENABLE** (default value): Row movement is enabled.

  – **DISABLE**: Row movement is disabled.

  If the row movement is enabled, an error may be reported when UPDATE and DELETE operations are performed concurrently. The causes are as follows:

  The old data is marked as deleted in the UPDATE and DELETE operations. If the row movement is enabled, the cross-partition update occurs when the partition key is updated, the kernel marks the old data in the old partition as

deleted and adds a data to the new partition. As a result, the new data cannot be found by querying the old data.

If data in the same row is concurrently operated, the cross-partition and non-cross-partition data results have different behaviors in the following three concurrency scenarios: UPDATE and UPDATE concurrency, DELETE and DELETE concurrency, UPDATE and DELETE concurrency.

a. For non-cross-partition data, no error is reported for the second operation after the first operation is performed.
   - If the first operation is UPDATE, the latest data can be found and operated after the second operation is performed.
   - If the first operation is DELETE, the second operation is terminated if the current data is deleted and the latest data cannot be found.

b. For the cross-partition data result, an error is reported for the second operation after the first operation is performed.
   - If the first operation is UPDATE, the second operation cannot find the latest data because the new data is in the new partition. Therefore, the second operation fails and an error is reported.
   - If the first operation is DELETE, performing the second operation can find that the current data is deleted and the latest data cannot be found, but cannot determine whether the operation of deleting the old data is UPDATE or DELETE. If the operation is UPDATE, an error is reported. If the operation is DELETE, the operation is terminated. To ensure the data correctness, an error is reported.

If the UPDATE and UPDATE concurrency, and UPDATE and DELETE concurrency are performed, the error can be solved only when the operations are performed serially. If the DELETE and DELETE concurrency are performed, the error can be solved by disabling the row movement.

- **NOT NULL**

  Forbids NULL values in columns. **ENABLE** can be omitted.

- **NULL**

  Allows to contain NULL values. This is the default setting.

  This clause is only provided for compatibility with non-standard SQL databases. It is not recommended.

- **CHECK (condition) [NO INHERIT]**

  Specifies an expression producing a Boolean result where the INSERT or UPDATE operation of new or updated rows can succeed only when the expression result is **TRUE** or **UNKNOWN**; otherwise, an error is thrown and the database is not altered.

  A check constraint specified as a column constraint should reference only the column's values, while an expression in a table constraint can reference multiple columns.

  A constraint marked with **NO INHERIT** will not propagate to child tables.

  ENABLE can be omitted.

- **DEFAULT default_expr**

  Assigns a default data value to a column. The value can be any variable-free expressions. (Subqueries and cross-references to other columns in the current

table are not allowed.) The data type of the default expression must match the data type of the column.

The default expression will be used in any insert operation that does not specify a value for the column. If there is no default value for a column, then the default value is **NULL**.

- **ON UPDATE update_expr**

  The ON UPDATE clause is an attribute constraint of a column.

  When an UPDATE operation is performed on a tuple in a table, if new values of updated columns are different from old values in the table, column values with this attribute but not in updated columns are automatically updated to the current timestamp. If new values of updated columns are the same as old values in the table, column values with this attribute but not in updated columns remain unchanged. If columns with this attribute are in updated columns, column values are updated according to the specified update value.

  NOTE

  - This attribute can be specified only in B-compatible database 5.7 (that is, **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**).
  - In terms of syntax, **update_expr** supports three keywords: CURRENT_TIMESTAMP, LOCALTIMESTAMP, and NOW(). You can also specify or not specify the precision of a keyword with parentheses. For example, ON UPDATE CURRENT_TIMESTAMP(), ON UPDATE CURRENT_TIMESTAMP(5), ON UPDATE LOCALTIMESTAMP(), and ON UPDATE LOCALTIMESTAMP(6). If the keyword does not contain parentheses or contains empty parentheses, the precision is 0. The NOW keyword cannot contain parentheses. The three types of keywords are synonyms of each other and have the same attribute effect.
  - This attribute can be specified only for columns of the following types: timestamp, datetime, date, time without time zone, smalldatetime, and abstime.
  - The CREATE TABLE AS syntax does not inherit the column attributes.
  - The CREATE TABLE LIKE syntax can use INCLUDING UPDATE or EXCLUDING UPDATE to inherit or exclude a constraint. The LIKE syntax is inherited from the LIKE syntax of PostgreSQL. Currently, the ILM policy information of the old table cannot be copied.
  - The precision specified by this attribute can be different from the precision specified by the type in the corresponding column. After the column value is updated through this attribute, the minimum precision is displayed. For example, CREATE TABLE t1 (col1 timestamp(6) ON UPDATE CURRENT_TIMESTAMP(3));. If the UPDATE syntax triggers the attribute to take effect, three decimal places in the value of **col1** are displayed after the update.
  - The same column cannot be specified for this attribute and the generated column constraint at the same time.
  - This attribute cannot be specified for the partition key in a partitioned table.

- **GENERATED ALWAYS AS (generation_expr) [STORED]**

  This clause creates a column as a generated column. The value of the generated column is calculated by **generation_expr** when data is written (inserted or updated). **STORED** indicates that the value of the generated column is stored as a common column.

NOTE

- The STORED keyword can be omitted, which has the same semantics as not omitting STORED.
- The generation expression cannot refer to data other than the current row in any way. The generation expression cannot reference other generation columns or system columns. The generation expression cannot return a result set. No subquery, aggregate function, or window function can be used. The function called by the generation expression can only be an immutable function.
- Default values cannot be specified for generated columns.
- The generated column cannot be used as a part of the partition key.
- Do not specify the generated column and the CASCADE, SET NULL, and SET DEFAULT actions of the ON UPDATE constraint clause together. Do not specify the generated column and the SET NULL, and SET DEFAULT actions of the ON DELETE constraint clause together.
- The method of modifying and deleting generated columns is the same as that of common columns. Delete the common column that the generated column depends on. The generated column is automatically deleted. The type of the column on which the generated column depends cannot be changed.
- The generated column cannot be directly written. In the INSERT or UPDATE statement, values cannot be specified for generated columns, but the keyword DEFAULT can be specified.
- The permission control for generated columns is the same as that for common columns.

- **GENERATED [ALWAYS | BY DEFAULT [ON NULL]] AS IDENTITY [(identity_options)]**

  Creates a column as an IDENTITY column. An implicit sequence is automatically created based on identity_options and attached to a specified column. When data is inserted, the value obtained from the sequence is automatically assigned to the column.

  – **GENERATED ALWAYS AS IDENTITY**: Only identity values provided by the sequence generator can be inserted into this column. User-specified values cannot be inserted into this column.

  – **GENERATED BY DEFAULT AS IDENTITY**: User-specified values are preferentially inserted into this column. If no value is specified, the identity values provided by the sequence generator are inserted.

  – **GENERATED BY DEFAULT ON NULL AS IDENTITY**: User-specified values are preferentially inserted into this column. If a null value is specified or no value is specified, the identity values provided by the sequence generator are inserted.

  The optional identity_options clause can be used to override sequence options.

  – **increment**: specifies the implicit sequence step. If the value is a positive number, an ascending sequence is generated. If the value is a negative number, a descending sequence is generated. The default value is **1**.

  – **MINVALUE minvalue | NO MINVALUE | NOMINVALUE**: specifies the minimum value of an execution sequence. If **minvalue** is not specified or **NO MINVALUE** is specified, the default value of an ascending sequence is **1**, and the default value of a descending sequence is **$-10^{27} + 1$**. **NOMINVALUE** is equivalent to **NO MINVALUE**.

  – **MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE**: specifies the maximum value of an execution sequence. If **maxvalue** is not specified or

NO MAXVALUE is specified, the default value of an ascending sequence is $10^{28} - 1$, and the default value of a descending sequence is **–1**. **NOMAXVALUE** is equivalent to **NO MAXVALUE**.

- **start** specifies the start value of an implicit sequence. The default value for an ascending sequence is **minvalue** and that for a descending sequence is **maxvalue**.
- **cache**: specifies the number of sequences stored in the memory for quick access purposes. The default value is **1**, indicating that only one value can be generated at a time, that is, no cache is generated.
- **NOCACHE**: No value of the sequence is stored in advance.
- **CYCLE**: recycles sequences after the number of sequences reaches **maxvalue** or **minvalue**. If **NO CYCLE** is declared, any call to nextval returns an error after the sequence reaches its maximum or minimum value. **NOCYCLE** is equivalent to **NO CYCLE**. The default value is **NO CYCLE**.
- **SCALE**: enables sequence scalability. If specified, a numeric offset is appended to the beginning of the sequence to prevent duplicate items in the generated value. If **NOSCALE** is declared, sequence scalability is disabled. The default value is **NOSCALE**.
- **EXTEND**: extends the numeric offset length (default value: **6**), aligns the sequence generation value to **x** (default value: **6**) + **y** (maximum number of digits). **SCALE** must be specified when **EXTEND** is specified. If **NOEXTEND** is declared, the numeric offset length is not extended. The default value is **NOEXTEND**.

NOTE

- The IDENTITY column can only be of the smallint, integer, bigint, decimal, numeric, float, double precision, or real type.
- In A-compatible mode, when an IDENTITY column of the integer type is created, the IDENTITY column is of the numeric type by default.
- The method of changing an IDENTITY column is the same as that of changing a common column, but the type can be changed only to smallint, integer, bigint, decimal, numeric, float, double precision, or real.
- By default, the IDENTITY column has the NOT NULL constraint.
- A table can contain only one IDENTITY column.
- The method of deleting an IDENTITY column is the same as that of deleting a common column. When a column is deleted, the implicit sequence of the IDENTITY column is automatically deleted.
- The IDENTITY column cannot be specified together with the SET DEFAULT action.
- The type of the implicit sequence that is automatically created is LARGE SEQUENCE.
- You cannot use DROP LARGE SEQUENCE or ALTER LARGE SEQUENCE to modify the implicit sequence of an identity.
- After permission is granted to the table, data can be inserted properly. To modify the IDENTITY column, delete the IDENTITY attribute, or delete the IDENTITY column, you need to grant permission to the corresponding implicit sequence.
- The [SCALE [EXTEND | NOEXTED] | NOSCALE] clause is available only when an IDENTITY column is created in a centralized system in A-compatible mode.
- In a fully-encrypted database, the encrypted IDENTITY column cannot be specified during table creation.

- **AUTO_INCREMENT**

  Specifies an auto-increment column.

  For details, see **AUTO_INCREMENT**.

- **UNIQUE [KEY] index_parameters**

  UNIQUE (column_name [, ...]) index_parameters

  Specifies that a group of one or more columns of a table can contain only unique values.

  For the purpose of a unique constraint, null is not considered equal.

  UNIQUE KEY can be used only when **sql_compatibility** is set to **'B'**, which has the same semantics as UNIQUE.

- **PRIMARY KEY index_parameters**

  PRIMARY KEY (column_name [, ...]) index_parameters

  Specifies that a column or columns of a table can contain only unique (non-duplicate) and non-null values.

  Only one primary key can be specified for a table.

- **DEFERRABLE | NOT DEFERRABLE**

  Determines whether the constraint can be deferred. A constraint that is not deferrable will be checked immediately after every command. Checking of constraints that are deferrable can be postponed until the end of the transaction using the **SET CONSTRAINTS** command. **NOT DEFERRABLE** is the default value. Currently, only UNIQUE constraints, primary key constraints, and foreign key constraints accept this clause. All the other constraints are not deferrable.

- **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

  If a constraint is deferrable, this clause specifies the default time to check the constraint.

  – If the constraint is **INITIALLY IMMEDIATE** (default value), it is checked after each statement.

  – If the constraint is **INITIALLY DEFERRED**, it is checked only at the end of the transaction.

  The constraint check time can be altered using the **SET CONSTRAINTS** statement.

- **USING INDEX TABLESPACE tablespace_name**

  Allows selection of the tablespace in which the index associated with a **UNIQUE** or **PRIMARY KEY** constraint will be created. If not specified, the index is created in **default_tablespace**. If **default_tablespace** is empty, the default tablespace of the database is used.

Examples

- Creating a level-2 partitioned table

```
-- Create a level-2 partitioned table tbl_list_list. Both of the level-1 and level-2 partitions are of the LIST type.
gaussdb=# CREATE TABLE tbl_list_list(
    sal_year    varchar(4)  NOT NULL,
    area_id     char(5)     NOT NULL,
    emp_id      char(5)     NOT NULL,
    sales_amt   int
) PARTITION BY LIST (sal_year) SUBPARTITION BY LIST(area_id)(
```



```
    PARTITION P_2019 VALUES ('2019')(
        SUBPARTITION p_2019_01001 VALUES ('01001'),
        SUBPARTITION p_2019_01002 VALUES ('01002'),
        SUBPARTITION p_2019_01003 VALUES ('01003')
    ),
    PARTITION p_2020 VALUES ('2020')(
        SUBPARTITION p_2020_01001 VALUES ('01001'),
        SUBPARTITION p_2020_01002 VALUES ('01002'),
        SUBPARTITION p_2020_01003 VALUES ('01003')
    )
);

-- Create a level-2 partitioned table tbl_range_list. The level-1 partition is of the RANGE type and
level-2 partition is of the LIST type.
gaussdb=# CREATE TABLE tbl_range_list(
    sal_date    varchar(6)  NOT NULL,
    area_id     char(5)     NOT NULL,
    emp_id      char(5)     NOT NULL,
    sales_amt   int
) PARTITION BY RANGE (sal_date) SUBPARTITION BY LIST(area_id)(
    PARTITION p_201901 VALUES LESS THAN (201902)(
        SUBPARTITION p_201901_01001 VALUES ('01001'),
        SUBPARTITION p_201901_01002 VALUES ('01002'),
        SUBPARTITION p_201901_01003 VALUES ('01003')
    ),
    PARTITION p_201902 VALUES LESS THAN (201903)(
        SUBPARTITION p_201902_01001 VALUES ('01001'),
        SUBPARTITION p_201902_01002 VALUES ('01002'),
        SUBPARTITION p_201902_01003 VALUES ('01003')
    )
);
```

- Specifying partitions in a level-2 partitioned table using DML statements
  - INSERT

    ```
    -- Insert data to a specified level-1 partition.
    gaussdb=# INSERT INTO tbl_range_list PARTITION(p_201901) VALUES('201901', '01001', '0001',
    75000 );

    -- The actual partition is inconsistent with the specified partition. An error is reported.
    gaussdb=# INSERT INTO tbl_range_list PARTITION(p_201902) VALUES('201901', '01001', '0002',
    6000);
    ERROR:  inserted partition key does not map to the table partition
    DETAIL:  N/A.

    -- Insert data to a specified level-2 partition.
    gaussdb=# INSERT INTO tbl_range_list SUBPARTITION(p_201902_01001) VALUES('201902',
    '01001', '0002', 8000);
    ```

  - SELECT

    ```
    -- Query data in a specified partition.
    gaussdb=# SELECT * FROM tbl_range_list PARTITION(p_201902);
     sal_date | area_id | emp_id | sales_amt
    ----------+---------+--------+-----------
     201902   | 01001   | 0002   |      8000
    (1 row)

    gaussdb=# SELECT * FROM tbl_range_list SUBPARTITION(p_201901_01001);
     sal_date | area_id | emp_id | sales_amt
    ----------+---------+--------+-----------
     201901   | 01001   | 0001   |     75000
    (1 row)
    ```

  - UPDATE

    ```
    -- Update data in a specified partition.
    gaussdb=# UPDATE tbl_range_list PARTITION(p_201901) SET sales_amt = 7000;

    gaussdb=# SELECT * FROM tbl_range_list;
     sal_date | area_id | emp_id | sales_amt
    ----------+---------+--------+-----------
     201901   | 01001   | 0001   |      7000
    ```



```
 201902   | 01001   | 0002   |      8000
(2 rows)

gaussdb=# UPDATE tbl_range_list SUBPARTITION FOR('201902','01001') SET sales_amt=6000;

gaussdb=# SELECT * FROM tbl_range_list;
 sal_date | area_id | emp_id | sales_amt
----------+---------+--------+-----------
 201901   | 01001   | 0001   |      7000
 201902   | 01001   | 0002   |      6000
(2 rows)
```

- DELETE

```
-- Delete data from a specified partition.
gaussdb=# DELETE FROM tbl_range_list PARTITION (p_201901);
DELETE 1

gaussdb=# DELETE FROM tbl_range_list SUBPARTITION (p_201902_01001);
DELETE 1

gaussdb=# DELETE FROM tbl_range_list SUBPARTITION for ('201901','01002');
DELETE 0

-- When sql_compatibility is set to 'B', data can be deleted from multiple partitions.
gaussdb=# CREATE DATABASE db dbcompatibility 'B';
gaussdb=# \c db
db=# CREATE TABLE range_list
(
   month_code VARCHAR2 ( 30 ) NOT NULL ,
   dept_code  VARCHAR2 ( 30 ) NOT NULL ,
   user_no    VARCHAR2 ( 30 ) NOT NULL ,
   sales_amt  int
)
PARTITION BY RANGE (month_code) SUBPARTITION BY LIST (dept_code)
(
  PARTITION p_201901 VALUES LESS THAN( '201903' )
  (
    SUBPARTITION p_201901_a VALUES ('1'),
    SUBPARTITION p_201901_b VALUES ('2')
  ),
  PARTITION p_201902 VALUES LESS THAN( '201910' )
  (
    SUBPARTITION p_201902_a VALUES ('1'),
    SUBPARTITION p_201902_b VALUES ('2')
  )
);

db=# DELETE FROM range_list AS t partition (p_201901_a, p_201901);
DELETE 0

-- Delete the database (replace the database name with the actual one).
db=# \c postgres
gaussdb=# DROP DATABASE db;

-- Delete the table.
gaussdb=# DROP TABLE tbl_list_list;
gaussdb=# DROP TABLE tbl_range_list;
```

7.12.8.50 CREATE TRIGGER

Description

CREATE TRIGGER is used to create a trigger. A trigger is a special type of stored procedure, and is used for complex service rules and requirements and help ensure reference integrity. A trigger is associated with a specified table or view and executes a specified function under specific conditions. This allows you to retain

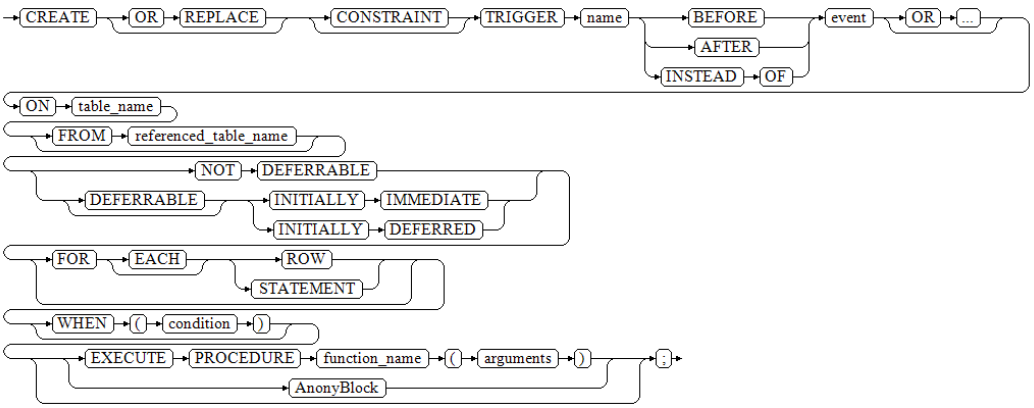
the relationships defined in the table when you add, update, or delete rows in the table.

Precautions

- Currently, triggers can be created only on ordinary row-store tables, instead of on temporary tables or unlogged tables.
- If multiple triggers of the same kind are defined for the same event, they will be fired in alphabetical order by name.
- Triggers are usually used for data association and synchronization between multiple tables. SQL execution performance is greatly affected. Therefore, you are advised not to use this statement when a large amount of data needs to be synchronized and performance requirements are high.
- When a trigger statement is executed, the permission is determined by the trigger creator.
- To create a trigger, you must have the TRIGGER permission on the specified table or have the CREATE ANY TRIGGER permission.
- A row-level trigger function triggered by BEFORE can return a **NULL** value, indicating that operations on the row are ignored. Subsequent triggers will not be executed and no INSERT, UPDATE, or DELETE action will be generated on the row. The return value of the trigger function triggered by AFTER is not affected.
- For a DELETE BEFORE trigger, the return value **NEW** indicates **NULL**. For an INSERT BEFORE trigger, the return value **OLD** indicates **NULL**. For an UPDATE BEFORE trigger, the return value of the trigger function is **NULL** only when it is displayed as **NULL**.
- For a trigger function whose event is INSERT or UPDATE, the normal return value is **NEW**. If a non-NULL row is returned, the inserted or updated row is modified. For a trigger function whose event is DELETE, the normal return value is **OLD**.
- INSTEAD OF triggers can only work on views. Their trigger functions can also return **NULL**, indicating that subsequent triggers will not be executed.

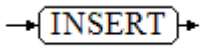
Syntax

```
CREATE [OR REPLACE] [ CONSTRAINT ] TRIGGER name { BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }
   ON table_name
   [ FROM referenced_table_name ]
   { NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE | INITIALLY DEFERRED } }
   [ FOR [ EACH ] { ROW | STATEMENT } ]
   [ WHEN ( condition ) ]
   [ EXECUTE PROCEDURE function_name ( arguments ) | AnonyBlock ];
```

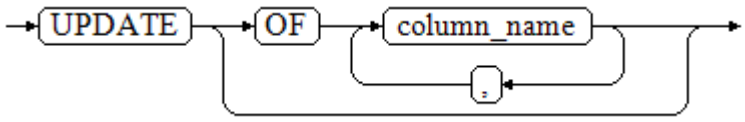



Events include:

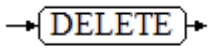
INSERT



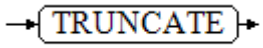
UPDATE [OF column_name [, ...]]



DELETE



TRUNCATE



Parameters

- **OR REPLACE**

  (Optional) If this parameter is specified, the existing trigger will be modified.

  Constraint triggers and internal triggers cannot be created or modified using the OR REPLACE syntax. A constraint trigger is a trigger created using CREATE CONSTRAINT TRIGGER. An internal trigger is a trigger implicitly created by some SQL statements. For example, if a foreign key constraint is added to a table, four triggers are implicitly created. The four triggers are internal triggers.

- **CONSTRAINT**

  (Optional) Creates a constraint trigger. That is, the trigger is used as a constraint. This is the same as a regular trigger except that the timing of the trigger firing can be adjusted using SET CONSTRAINTS. Constraint triggers must be AFTER ROW triggers.

- **name**

  Specifies the name of the trigger to be created. This must be distinct from the name of any other trigger for the same table. The name cannot be schema-

qualified — the trigger inherits the schema of its table. For a constraint trigger, this is also the name to use when modifying the trigger's behavior using **SET CONSTRAINTS**.

Value range: a string, which complies with the **naming convention**. A value can contain a maximum of 63 characters.

- **BEFORE**

  Specifies that the function is called before the event.

- **AFTER**

  Specifies that the function is called after the event. A constraint trigger can only be specified as **AFTER**.

- **INSTEAD OF**

  Specifies that the function is called instead of the event.

- **event**

  Specifies the event that will fire the trigger. Values are **INSERT**, **UPDATE**, **DELETE**, and **TRUNCATE**. Multiple events can be specified using **OR**.

  For UPDATE events, it is possible to specify a list of columns using this syntax:

  ```
  UPDATE OF column_name1 [, column_name2 ... ]
  ```

  The trigger will only fire if at least one of the listed columns is mentioned as a target of the UPDATE statement. INSTEAD OF UPDATE events do not support lists of columns. If the column specified by UPDATE OF contains a generated column, the trigger is also fired when the column on which the generated column depends is the target column of the UPDATE statement.

- **table_name**

  Specifies the name of the table for which the trigger is created.

  Value range: name of an existing table in the database

- **referenced_table_name**

  Specifies the name of another table referenced by the constraint. This option is used for foreign-key constraints. It can only be specified for constraint triggers. Because foreign keys are not supported currently, this option is not recommended for general use.

  Value range: name of an existing table in the database

- **DEFERRABLE | NOT DEFERRABLE**

  Specifies the start time of the trigger. It can only be specified for constraint triggers. Determines whether the constraint can be deferred.

  For details, see **CREATE TABLE**.

- **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

  If a constraint is deferrable, this clause specifies the default time to check the constraint. It can only be specified for constraint triggers.

  For details, see **CREATE TABLE**.

- **FOR EACH ROW | FOR EACH STATEMENT**

  Specifies the frequency of firing the trigger.

  - **FOR EACH ROW** indicates that the trigger should be fired once for every row affected by the trigger event.
  - **FOR EACH STATEMENT** indicates that the trigger should be fired just once per SQL statement.

If neither is specified, the default is **FOR EACH STATEMENT**. Constraint triggers can only be marked as **FOR EACH ROW**.

- **condition**

  Specifies whether the trigger function will actually be executed. If **WHEN** is specified, the function will be called only when **condition** returns **true**.

  In FOR EACH ROW triggers, the WHEN condition can refer to columns of the old and/or new row values by writing **OLD.***column name* or **NEW.***column name* respectively. In addition, INSERT triggers cannot refer to OLD, and DELETE triggers cannot refer to NEW.

  INSTEAD OF triggers do not support WHEN conditions.

  Currently, WHEN expressions cannot contain subqueries.

  Note that for constraint triggers, evaluation of the WHEN condition is not deferred, but occurs immediately after the row update operation is performed. If the condition does not evaluate to **true**, then the trigger is not queued for deferred execution.

- **ANONYBLOCK**

  For the anonymous block in an A-compatible database, statements are executed when a trigger is triggered. When you use the anonymous block syntax to create a trigger, a function with the same name as the trigger is created. Ensure that the function name does not conflict with any existing function name.

- **function_name**

  Specifies a user-defined function, which must be declared as taking no parameters and returning type trigger. This is executed when a trigger fires.

- **arguments**

  Specifies an optional comma-separated list of parameters to be provided to the function when the trigger is executed. The parameters are literal string constants. Simple names and numeric constants can also be written here, but they will all be converted to strings. Check the description of the implementation language of the trigger function to find out how these parameters can be accessed within the function.

  NOTE

  The following details trigger types:

  - INSTEAD OF triggers must be marked as FOR EACH ROW and can be defined only on views.
  - BEFORE and AFTER triggers on a view must be marked as FOR EACH STATEMENT.
  - TRUNCATE triggers must be marked as FOR EACH STATEMENT.

Table 7-221 Types of triggers supported on tables and views

When	Event	Row-Level	Statement-Level
BEFORE	INSERT/UPDATE/DELETE	Tables	Tables and views
	TRUNCATE	Not supported.	Tables

When	Event	Row-Level	Statement-Level
AFTER	INSERT/UPDATE/DELETE	Tables	Tables and views
	TRUNCATE	Not supported.	Tables
INSTEAD OF	INSERT/UPDATE/DELETE	Views	Not supported.
	TRUNCATE	Not supported.	Not supported.

Table 7-222 Special variables of PL/pgSQL trigger functions

Variable	Description
NEW	New tuple for INSERT and UPDATE operations. This variable is **NULL** for **DELETE** operations and statement-level triggers.
OLD	Old tuple for UPDATE and DELETE operations. This variable is **NULL** for **INSERT** operations and statement-level triggers.
TG_NAME	Trigger name.
TG_WHEN	Trigger timing (**BEFORE**, **AFTER**, or **INSTEAD OF**).
TG_LEVEL	Trigger frequency (**ROW** or **STATEMENT**).
TG_OP	Trigger operations (**INSERT**/**UPDATE**/**DELETE**/**TRUNCATE**) that must be capitalized.
TG_RELID	OID of the table where the trigger resides.
TG_RELNAME	Name of the table where the trigger resides. (This variable has been replaced by **TG_TABLE_NAME**.)
TG_TABLE_NAME	Name of the table where the trigger resides.
TG_TABLE_SCHEMA	Schema of the table where the trigger resides.
TG_NARGS	Number of parameters for the trigger function.

Variable	Description
TG_ARGV[]	List of parameters for the trigger function.

Examples

```
-- Create a source table and a target table.
gaussdb=# CREATE TABLE test_trigger_src_tbl(id1 INT, id2 INT, id3 INT);
gaussdb=# CREATE TABLE test_trigger_des_tbl(id1 INT, id2 INT, id3 INT);
```

Use the INSERT trigger.

```
-- Create an INSERT trigger function.
gaussdb=# CREATE OR REPLACE FUNCTION tri_insert_func() RETURNS TRIGGER AS
        $$
        DECLARE
        BEGIN
                INSERT INTO test_trigger_des_tbl VALUES(NEW.id1, NEW.id2, NEW.id3);
                RETURN NEW;
        END
        $$ LANGUAGE plpgsql;

-- Create an INSERT trigger.
gaussdb=# CREATE TRIGGER insert_trigger
        BEFORE INSERT ON test_trigger_src_tbl
        FOR EACH ROW
        EXECUTE PROCEDURE tri_insert_func();

-- Execute the INSERT event and check the trigger results.
gaussdb=# INSERT INTO test_trigger_src_tbl VALUES(100,200,300);

gaussdb=# SELECT * FROM test_trigger_src_tbl;
 id1 | id2 | id3
-----+-----+-----
 100 | 200 | 300
(1 row)

gaussdb=# SELECT * FROM test_trigger_des_tbl; // Check whether the trigger operation takes effect.
 id1 | id2 | id3
-----+-----+-----
 100 | 200 | 300
(1 row)
```

Use anonymous blocks and the OR REPLACE syntax to create triggers.

```
-- Create an INSERT trigger using the anonymous block syntax.
gaussdb=# CREATE TRIGGER insert_trigger_with_anonyblock
        BEFORE INSERT ON test_trigger_src_tbl
        FOR EACH ROW
        DECLARE
        BEGIN
                INSERT INTO test_trigger_des_tbl VALUES(NEW.id1, NEW.id2, NEW.id3);
                RETURN NEW;
        END;
        /

-- Create an INSERT trigger using the OR REPLACE syntax.
gaussdb=# CREATE OR REPLACE TRIGGER insert_trigger_with_anonyblock
        BEFORE INSERT ON test_trigger_src_tbl
        FOR EACH ROW
        DECLARE
        BEGIN
                INSERT INTO test_trigger_des_tbl VALUES(NEW.id1, NEW.id2, NEW.id3);
                RETURN NEW;
        END;
```



```
        /

-- Delete the trigger.
gaussdb=# DROP TRIGGER insert_trigger_with_anonyblock ON test_trigger_src_tbl; -- Delete the implicitly created function insert_trigger_with_anonyblock.
```

Use the UPDATE trigger.

```
-- Create an UPDATE trigger function.
gaussdb=# CREATE OR REPLACE FUNCTION tri_update_func() RETURNS TRIGGER AS
$$
DECLARE
BEGIN
UPDATE test_trigger_des_tbl SET id3 = NEW.id3 WHERE id1=OLD.id1;
RETURN OLD;
END
$$ LANGUAGE plpgsql;

-- Create an UPDATE trigger.
gaussdb=# CREATE TRIGGER update_trigger
        AFTER UPDATE ON test_trigger_src_tbl
        FOR EACH ROW
        EXECUTE PROCEDURE tri_update_func();

-- Execute the UPDATE event and check the trigger results.
gaussdb=# UPDATE test_trigger_src_tbl SET id3=400 WHERE id1=100;

gaussdb=# SELECT * FROM test_trigger_src_tbl;
 id1 | id2 | id3
-----+-----+-----
 100 | 200 | 400
(1 row)

gaussdb=# SELECT * FROM test_trigger_des_tbl; // Check whether the trigger operation takes effect.
 id1 | id2 | id3
-----+-----+-----
 100 | 200 | 400
(1 row)
```

Use the DELETE trigger.

```
-- Create a DELETE trigger function.
gaussdb=# CREATE OR REPLACE FUNCTION TRI_DELETE_FUNC() RETURNS TRIGGER AS
        $$
        DECLARE
        BEGIN
                DELETE FROM test_trigger_des_tbl WHERE id1=OLD.id1;
                RETURN OLD;
        END
        $$ LANGUAGE plpgsql;

-- Create a DELETE trigger.
gaussdb=# CREATE TRIGGER delete_trigger BEFORE DELETE ON test_trigger_src_tbl FOR EACH ROW
EXECUTE PROCEDURE tri_delete_func();

-- Execute the DELETE event and check the trigger results.
gaussdb=# DELETE FROM test_trigger_src_tbl WHERE id1=100;

gaussdb=# SELECT * FROM test_trigger_src_tbl;
 id1 | id2 | id3
-----+-----+-----
(0 rows)

gaussdb=# SELECT * FROM test_trigger_des_tbl; // Check whether the trigger operation takes effect.
 id1 | id2 | id3
-----+-----+-----
(0 rows)
```

Rename a trigger.


```
-- Rename a trigger.
gaussdb=# ALTER TRIGGER delete_trigger ON test_trigger_src_tbl RENAME TO delete_trigger_renamed;
```

Disable a trigger.

```
-- Disable insert_trigger.
gaussdb=# ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER insert_trigger;

gaussdb=# INSERT INTO test_trigger_src_tbl VALUES(100,200,300);

gaussdb=# SELECT * FROM test_trigger_src_tbl;
 id1 | id2 | id3
-----+-----+-----
 100 | 200 | 300
(1 row)

gaussdb=# SELECT * FROM test_trigger_des_tbl; // The trigger does not take effect.
 id1 | id2 | id3
-----+-----+-----
(0 rows)

-- Disable all triggers on the current table.
gaussdb=# ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER ALL;
```

Delete a trigger.

```
gaussdb=# DROP TRIGGER insert_trigger ON test_trigger_src_tbl;

gaussdb=# DROP TRIGGER update_trigger ON test_trigger_src_tbl;

gaussdb=# DROP TRIGGER delete_trigger_renamed ON test_trigger_src_tbl;
```

Delete a function.

```
gaussdb=# DROP FUNCTION tri_insert_func;

gaussdb=# DROP FUNCTION tri_update_func;

gaussdb=# DROP FUNCTION tri_delete_func;
-- Delete the source table and target table.
gaussdb=# DROP TABLE test_trigger_src_tbl;
gaussdb=# DROP TABLE test_trigger_des_tbl;
```

Helpful Links

ALTER TRIGGER, **DROP TRIGGER**, and **ALTER TABLE**

7.12.8.51 CREATE TYPE

Description

Defines a new data type for use in the current database. The user who defines a type becomes its owner. Types are designed only for row-store tables.

The following data types can be created: composite type, base type, shell type, enumerated type, and set type.

- Composite type

  A composite type is specified by a list of attribute names and data types. If the data type of an attribute is collatable, the attribute's collation rule can also be specified. This is essentially the same as the row type of a table, but using **CREATE TYPE** avoids the need to create an actual table when all that is wanted is to define a type. A stand-alone composite type is useful as the parameter or return type of a function.

To create a composite type, you must have the **USAGE** permission on all of its attribute types.

- Base type

  You can create a base type (scalar type). Generally, these functions must be written in the underlying language.

- Shell type

  A shell type is simply a placeholder for a type to be defined later; it is created by issuing **CREATE TYPE** with no parameters except for the type name. Shell types are needed as forward references when base types are created.

- Enumerated type

  An enumerated type is a list of one or more quoted labels, each of which must be 1 to 63 bytes long.

- Set type

  It is similar to an array but has no length limit. It is mainly used in stored procedures.

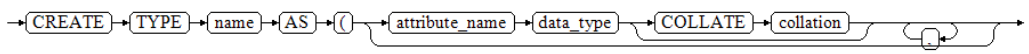
- A user granted with the **CREATE ANY TYPE** permission can create types in the public and user schemas.

Precautions

- If a schema name is given then the type is created in the specified schema. Otherwise, it is created in the current schema. The type name must be distinct from the name of any existing type or domain in the same schema. (Because tables have associated data types, the type name must also be distinct from the name of any existing table in the same schema.)
- When creating a non-system type by associating a function, the user needs to understand the definition of the type and the function associated with the type. If this function is not properly used, permissions may be exploited due to the associated function.

Syntax

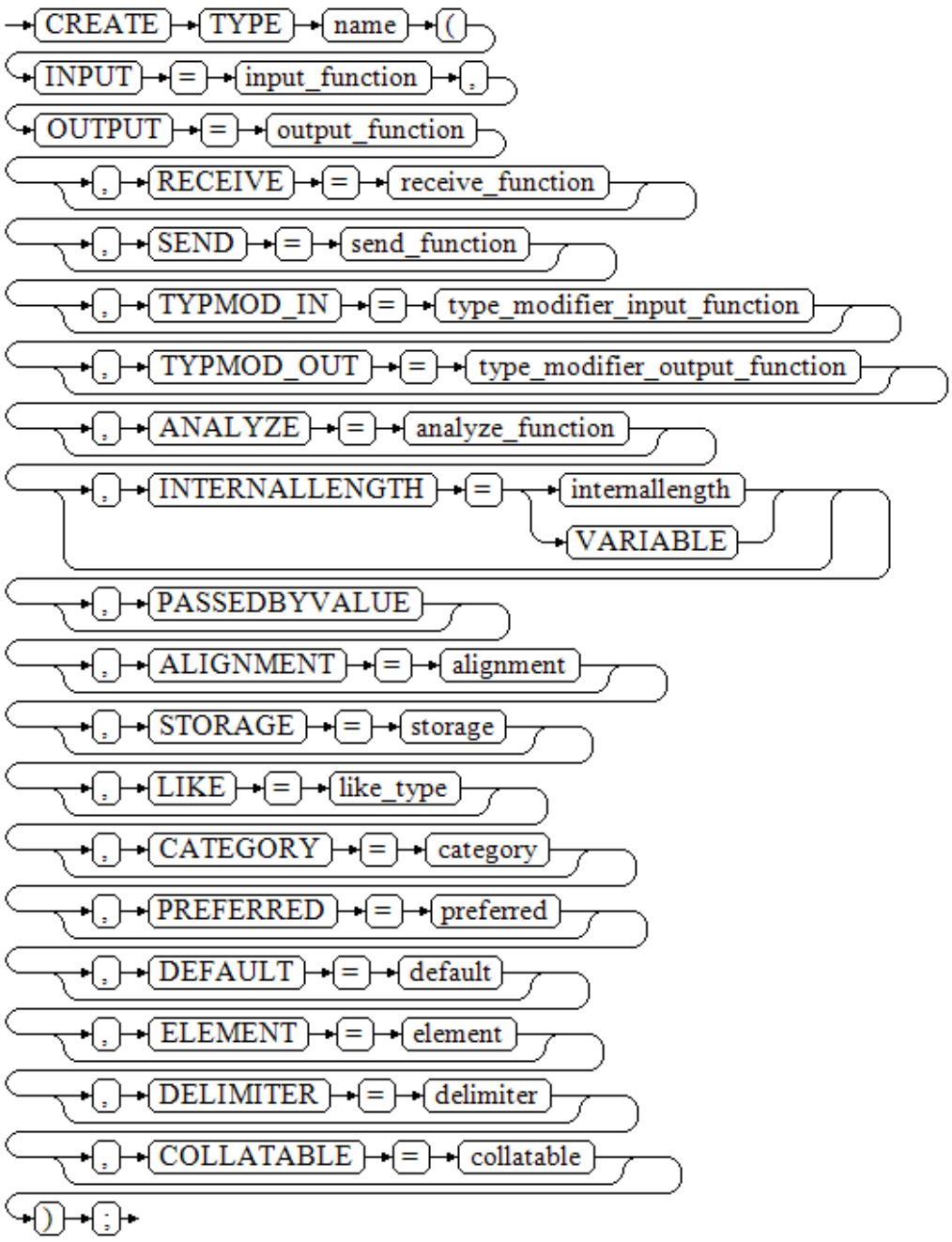
```
CREATE TYPE name AS
    ( [ attribute_name data_type [ COLLATE collation ] [, ... ] ] );
```



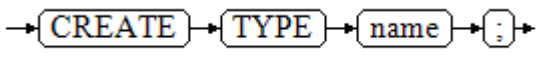

```
CREATE TYPE name (
    INPUT = input_function,
    OUTPUT = output_function
    [ , RECEIVE = receive_function ]
    [ , SEND = send_function ]
    [ , TYPMOD_IN = type_modifier_input_function ]
    [ , TYPMOD_OUT = type_modifier_output_function ]
    [ , ANALYZE = analyze_function ]
    [ , INTERNALLENGTH = { internallength | VARIABLE } ]
    [ , PASSEDBYVALUE ]
    [ , ALIGNMENT = alignment ]
    [ , STORAGE = storage ]
    [ , LIKE = like_type ]
    [ , CATEGORY = category ]
    [ , PREFERRED = preferred ]
    [ , DEFAULT = default ]
    [ , ELEMENT = element ]
    [ , DELIMITER = delimiter ]
```



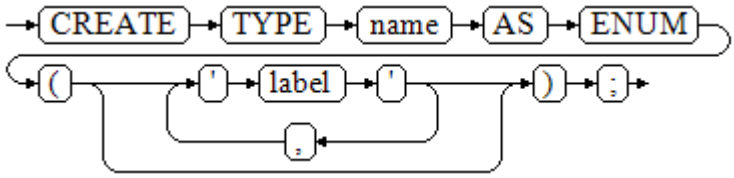
```
    [ , COLLATABLE = collatable ]
);
```



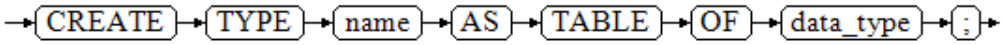

```
CREATE TYPE name;
```




```
CREATE TYPE name AS ENUM
    ( [ 'label' [, ... ] ] );
```


```
CREATE TYPE name AS TABLE OF data_type;
```



Parameters

Composite type

- **name**

  Specifies the name (optionally schema-qualified) of the type to be created.
- **attribute_name**

  Specifies the name of an attribute (column) for the composite type.
- **data_type**

  Specifies the name of an existing data type to become a column of the composite type. You can use **%ROWTYPE** to indirectly reference the type of a table, or **%TYPE** to indirectly reference the type of a column in a table or composite type.
- **collation**

  Specifies the name of an existing collation rule to be associated with a column of the composite type. You can run the **SELECT * FROM pg_collation** command to query collation rules from the pg_collation system catalog. The default collation rule is the row starting with **default** in the query result.

  NOTE

  The constructor of the composite type can use the => to assign values. The restrictions are as follows:

  - The => function is used when the constructor of the composite type assigns values. This function is supported only for A-compatible databases (**sql_compatibility** set to **'A'**).
  - The constructor of the composite type uses the => function to assign values. This function is supported only when => is consecutively used to assign values to input parameters, for example, **composite_name(elename1 => val1, elename2 => val2, elename3 => val3)** or **composite_name(val1, elename2 => val2, elename3 => val3)**. Discontinuous use of => or => is not supported. The last input parameter, for example, **composite_name(elename1 => val1, elename2 => val2, val3)** or **composite_name(val1, elename2 => val2, val3)**, is not assigned a value.

Base type

When creating a base type, you can place parameters in any order. The **input_function** and **output_function** parameters are mandatory, and other parameters are optional.

- **input_function**

  Specifies the name of a function that converts data from the type's external textual form to its internal form.

The input function may be declared as taking one parameter of type cstring or taking three parameters of types cstring, oid, and integer.

- The cstring-type parameter is the input text as a C string.
- The oid-type parameter is the type's own OID (except for array types, which instead receive their element type's OID).
- The integer-type parameter is typmod of the target column, if known (**–1** will be passed if not known).

The input function must return a value of the data type itself. Usually, an input function should be declared as **STRICT**. Otherwise, when a **NULL** input value is read and the input function is called, the first parameter is **NULL**. The function must still return **NULL** in this case, unless it raises an error. (This case is mainly meant to support domain input functions, which might need to reject **NULL** inputs.)

NOTE

- The input and output functions can be declared to have results or parameters of the new type, when they have to be created before the new type can be created. The type should first be defined as a shell type, which is a placeholder type that has no attributes except a name and an owner. This is done by running the **CREATE TYPE** *name* command, with no additional parameters. Then, the I/O functions written in C can be defined as referencing the shell type. Finally, CREATE TYPE with a full definition replaces the shell entry with a complete, valid type definition, after which the new type can be used normally.
- If the input and output functions are internal functions and are specified as internal system functions, the parameter types of the input and output functions must be the same, and the parameter types of **INTERNALLENGTH** and **PASSEDBYVALUE** of the new type must be the same as those of the input and output functions.

- **output_function**

  Specifies the name of a function that converts data from the type's internal form to its external textual form.

  The output function must be declared as taking one parameter of the new data type. The output function must return type cstring. Output functions are not called for **NULL** values.

- **receive_function**

  (Optional) Specifies the name of a function that converts data from the type's external binary form to its internal form.

  If this function is not supplied, the type cannot participate in binary input. The binary representation should be chosen to be cheap to convert to internal form, while being reasonably portable. (For example, the standard integer data types use network byte order as the external binary representation, while the internal representation is in the machine's native byte order.) The receive function should perform adequate checking to ensure that the value is valid.

  The receive function may be declared as taking one parameter of type internal or taking three parameters of types internal, oid, integer.

  - The internal-type parameter is a pointer to a StringInfo buffer holding the received byte strings.
  - The oid- and integer-type parameters are the same as those of the text input function.

  The receive function must return a value of the data type itself. Usually, a receive function should be declared as **STRICT**. If it is not, it will be called with

a **NULL** first parameter when reading a **NULL** input value. The function must still return **NULL** in this case, unless it raises an error. (This case is mainly meant to support domain receive functions, which might need to reject **NULL** inputs.)

- **send_function**

  (Optional) Specifies the name of a function that converts data from the type's internal form to its external binary form.

  If this function is not supplied, the type cannot participate in binary output. The send function must be declared as taking one parameter of the new data type. The send function must return type bytea. Send functions are not called for **NULL** values.

- **type_modifier_input_function**

  (Optional) Specifies the name of a function that converts an array of modifiers for a type to its internal format.

- **type_modifier_output_function**

  (Optional) Specifies the name of a function that converts the internal format of modifiers for a type to its external text format.

  NOTE

  type_modifier_input_function and **type_modifier_output_function** are needed if the type supports modifiers, that is optional constraints attached to a type declaration, such as char(5) or numeric(30,2). GaussDB allows user-defined types to take one or more simple constants or identifiers as modifiers. However, this information must be capable of being packed into a single non-negative integer value for storage in the system catalogs. Declared modifiers are passed to **type_modifier_input_function** in the cstring array format. It must check the values for validity (throwing an error if they are wrong), and if they are correct, return a single non-negative integer value that will be stored as the column "typmod". Type modifiers will be rejected if the type does not have a **type_modifier_input_function**. The **type_modifier_output_function** converts the internal integer typmod value back to the correct form for user display. It must return a cstring value that is the exact string to append to the type name. For example, a numeric function may return (30,2). It is allowed to omit the **type_modifier_output_function**, in which case the default display format is just the stored typmod integer value enclosed in parentheses.

- **analyze_function**

  (Optional) Specifies the name of a function that performs statistical analysis for the data type.

  By default, **ANALYZE** will attempt to gather statistics using the type's "equals" and "less-than" operators, if there is a default B-tree operator class for the type. For non-scalar types, this behavior is likely to be unsuitable, so it can be overridden by specifying a custom analysis function. The analysis function must be declared to take one parameter of type internal and return a boolean result.

- **internallength**

  (Optional) Specifies the length in bytes of the new type's internal representation. The default assumption is that it is variable-length.

  While the details of the new type's internal representation are only known to the I/O functions and other functions you create to work with the type, there are several attributes of the internal representation that must be declared to GaussDB. Foremost of these is **internallength**. Base data types can be fixed-length, in which case **internallength** is a positive integer, or variable length,

indicated by setting **internallength** to **VARIABLE**. (Internally, this is represented by setting **typlen** to **-1**.) The internal representation of all variable-length types must start with a 4-byte integer giving the total length of this value of the type.

- **PASSEDBYVALUE**

  (Optional) Indicates that values of this data type are passed by value, rather than by reference. You cannot pass by value types whose internal representation is larger than the size of the Datum type (4 bytes on most machines, 8 bytes on a few).

- **alignment**

  (Optional) Specifies the storage alignment requirement of the data type. If specified, it must be **char**, **int2**, **int4**, or **double**. The default is **int4**.

  The allowed values equate to alignment on 1, 2, 4, or 8 byte boundaries. Note that variable-length types must have an alignment of at least 4, since they necessarily contain an int4 as their first component.

- **storage**

  (Optional) Specifies the storage strategy for the data type.

  If specified, it must be **plain**, **external**, **extended**, or **main**. The default is **plain**.

  - **plain** specifies that data of the type will always be stored in-line and not compressed. (Only **plain** is allowed for fixed-length types.)
  - **extended** specifies that the system will first try to compress a long data value, and will move the value out of the main table row if it is still too long.
  - **external** allows the value to be moved out of the main table, but the system will not try to compress it.
  - **main** allows compression, but discourages moving the value out of the main table. (Data items with this storage strategy might still be moved out of the main table if there is no other way to make a row fit, but they will be kept in the main table preferentially over **extended** and **external** items.)

    All **storage** values other than **plain** imply that the functions of the data type can handle values that have been toasted. The specific other value given merely determines the default **TOAST** storage strategy for columns of a toastable data type; users can pick other strategies for individual columns using **ALTER TABLE SET STORAGE**.

- **like_type**

  (Optional) Specifies the name of an existing data type that the new type will have the same representation as. The values of **internallength**, **passedbyvalue**, **alignment**, and **storage** are copied from that type, unless overridden by explicit specification elsewhere in this **CREATE TYPE** statement.

  Specifying representation in this way is especially useful when the low-level implementation of a new type references an existing type.

- **category**

  (Optional) Specifies the category code (a single ASCII character) for this type. The default is **U** for a user-defined type. You may also choose other ASCII characters to create custom categories.

- **preferred**

  (Optional) Specifies whether a type is preferred within its type category. If it is, the value will be **TRUE**, else **FALSE**. The default is **FALSE**. Be very careful about creating a preferred type within an existing type category, as this could cause surprising changes in behavior.

  NOTE

  The **category** and **preferred** parameters can be used to help control which implicit cast will be applied in ambiguous situations. Each data type belongs to a category named by a single ASCII character, and each type is either preferred or not within its category. The parser will prefer casting to preferred types (but only from other types within the same category) when this rule is helpful in resolving overloaded functions or operators. For types that have no implicit casts to or from any other types, it is sufficient to leave these settings at the defaults. However, for a group of related types that have implicit casts, it is often helpful to mark them all as belonging to a category and select one or two of the most general types as being preferred within the category. The **category** parameter is especially useful when adding a user-defined type to an existing built-in category, such as the numeric or string types. However, it is also possible to create entirely-user-defined type categories. Select any ASCII character other than an uppercase letter to name such a category.

- **default**

  (Optional) Specifies the default value for the data type. If this is omitted, the default is null.

  A default value can be specified, in case a user wants columns of the data type to default to something other than the null value. Specify the default with the **DEFAULT** keyword. (Such a default can be overridden by an explicit **DEFAULT** clause attached to a particular column.)

- **element**

  (Optional) Specifies the type of array elements when an array type is created. For example, to define an array of 4-byte integers (int4), specify **ELEMENT = int4**.

- **delimiter**

  (Optional) Specifies the delimiter character to be used between values in arrays made of this type.

  delimiter can be set to a specific character. The default delimiter is the comma (,). Note that the delimiter is associated with the array element type, not the array type itself.

- **collatable**

  (Optional) Specifies whether this type's operations can use collation information. If they can, the value will be **TRUE**, else **FALSE** (default).

  If **collatable** is **TRUE**, column definitions and expressions of the type may carry collation information through use of the **COLLATE** clause. It is up to the implementations of the functions operating on the type to actually make use of the collation information; this does not happen automatically merely by marking the type collatable.

- **label**

  (Optional) Represents the textual label associated with one value of an enumerated type. It is a string of 1 to 63 characters.

NOTE

Whenever a user-defined type is created, GaussDB automatically creates an associated array type whose name consists of the element type's name prepended with an underscore (_).

Examples

```
-- Create a composite type, create a table, insert data, and make a query.
gaussdb=# CREATE TYPE compfoo AS (f1 int, f2 text);
gaussdb=# CREATE TABLE t1_compfoo(a int, b compfoo);
gaussdb=# CREATE TABLE t2_compfoo(a int, b compfoo);
gaussdb=# INSERT INTO t1_compfoo values(1,(1,'demo'));
gaussdb=# INSERT INTO t2_compfoo SELECT * FROM t1_compfoo;
gaussdb=# SELECT (b).f1 FROM t1_compfoo;
gaussdb=# SELECT * FROM t1_compfoo t1 JOIN t2_compfoo t2 ON (t1.b).f1=(t1.b).f1;

-- Rename the data type.
gaussdb=# ALTER TYPE compfoo RENAME TO compfoo1;

-- Change the owner of the user-defined type compfoo1 to usr1.
gaussdb=# CREATE USER usr1 PASSWORD '********';
gaussdb=# ALTER TYPE compfoo1 OWNER TO usr1;

-- Change the schema of the user-defined type compfoo1 to usr1.
gaussdb=# ALTER TYPE compfoo1 SET SCHEMA usr1;

-- Add a new attribute to the data type.
gaussdb=# ALTER TYPE usr1.compfoo1 ADD ATTRIBUTE f3 int;

-- Delete the compfoo1 type.
gaussdb=# DROP TYPE usr1.compfoo1 CASCADE;

-- Delete related tables and users.
gaussdb=# DROP TABLE t1_compfoo;
gaussdb=# DROP TABLE t2_compfoo;
gaussdb=# DROP SCHEMA usr1;
gaussdb=# DROP USER usr1;

-- Create an enumerated type.
gaussdb=# CREATE TYPE bugstatus AS ENUM ('create', 'modify', 'closed');

-- Add a label.
gaussdb=# ALTER TYPE bugstatus ADD VALUE IF NOT EXISTS 'regress' BEFORE 'closed';

-- Rename a label.
gaussdb=# ALTER TYPE bugstatus RENAME VALUE 'create' TO 'new';

-- Create a set type.
gaussdb=# CREATE TYPE bugstatus_table AS TABLE OF bugstatus;

-- Delete the collection type and enumerated type.
gaussdb=# DROP TYPE bugstatus_table;
gaussdb=# DROP TYPE bugstatus CASCADE;

-- Create an A-compatible database and switch to this database.
gaussdb=#  CREATE DATABASE ora_compatible_db DBCOMPATIBILITY 'A';
CREATE DATABASE
gaussdb=#  \c ora_compatible_db;
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "ora_compatible_db" as user "omm".
-- Create a composite type.
ora_compatible_db=# CREATE TYPE person_type AS  (name VARCHAR2(50), age NUMBER, gender
VARCHAR2(10));
CREATE TYPE
ora_compatible_db=# CREATE TYPE address_type AS  (street VARCHAR2(50), city VARCHAR2(50), zip_code
VARCHAR2(10));
CREATE TYPE
```



```
ora_compatible_db=# CREATE TYPE customer_type AS  (id NUMBER, person_info person_type, address_info
address_type);
CREATE TYPE
-- When the constructor of the composite type assigns values, all input parameters use =>. The execution is
successful.
ora_compatible_db=# DECLARE
  v_customer customer_type;
 BEGIN
  v_customer := customer_type(
    id => 123,
    person_info => person_type(name => 'John', age => 30, gender => 'Male'),
    address_info => address_type(street => '123 Main St', city => 'Anytown', zip_code => '12345')
  );
 END;
 /
ANONYMOUS BLOCK EXECUTE
-- When the constructor of the composite type assigns values, use => to assign values to the last input
parameter. The execution is successful.
ora_compatible_db=# DECLARE
  v_person person_type;
 BEGIN
  v_person := person_type('John', age => 30, gender => 'Male');
 END;
 /
ANONYMOUS BLOCK EXECUTE
-- The constructor of the composite type does not consecutively use => to assign values to the last input
parameter. As a result, an error message is displayed.
ora_compatible_db=# DECLARE
  v_person person_type;
 BEGIN
  v_person := person_type(name => 'John', age => 30, 'Male');
 END;
 /
ERROR:  positional argument cannot follow named argument
LINE 1: SELECT person_type(name => 'John', age => 30, 'Male')
                                                      ^
QUERY:  SELECT person_type(name => 'John', age => 30, 'Male')
CONTEXT:  referenced column: person_type
PL/pgSQL function inline_code_block line 3 at assignment
-- Delete the created database and switch back to the original database.
ora_compatible_db=# \c postgres;
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "postgres" as user "omm".
gaussdb=# DROP DATABASE ora_compatible_db;
DROP DATABASE
```

Helpful Links

ALTER TYPE and **DROP TYPE**

7.12.8.52 CREATE USER

Description

Creates a user with a specified password. A user is a basic element for GaussDB authentication. You can use the correct username and password to log in to the GaussDB and grant different permissions to different users so that different users can perform different operations.

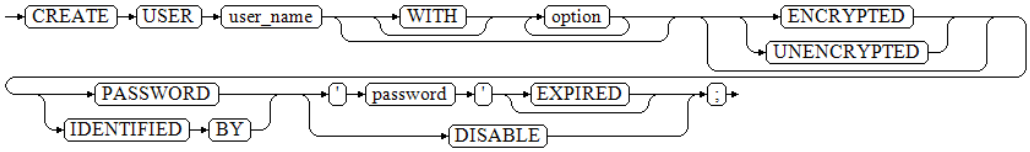
Precautions

- A user created using the CREATE USER statement has the LOGIN permission by default.

- When you run the **CREATE USER** command to create a user, the system creates a schema with the same name as the user in the database where the command is executed.
- The owner of an object created by a system administrator in a schema with the same name as a common user is the common user, not the system administrator.

Syntax

```
CREATE USER user_name [ [ WITH ] option [ ... ] ] [ ENCRYPTED | UNENCRYPTED ] { PASSWORD |
IDENTIFIED BY } { 'password' [EXPIRED] | DISABLE };
```



The option clause is used to configure information, including permissions and properties.

```
{SYSADMIN | NOSYSADMIN}
    | {MONADMIN | NOMONADMIN}
    | {OPRADMIN | NOOPRADMIN}
    | {POLADMIN | NOPOLADMIN}
    | {AUDITADMIN | NOAUDITADMIN}
    | {CREATEDB | NOCREATEDB}
    | {USEFT | NOUSEFT}
    | {CREATEROLE | NOCREATEROLE}
    | {INHERIT | NOINHERIT}
    | {LOGIN | NOLOGIN}
    | {REPLICATION | NOREPLICATION}
    | {PERSISTENCE | NOPERSISTENCE}
    | CONNECTION LIMIT connlimit
    | VALID BEGIN 'timestamp'
    | VALID UNTIL 'timestamp'
    | RESOURCE POOL 'respool'
    | USER GROUP 'groupuser'
    | PERM SPACE 'spacelimit'
    | TEMP SPACE 'tmpspacelimit'
    | SPILL SPACE 'spillspacelimit'
    | NODE GROUP logic_cluster_name
    | IN ROLE role_name [, ...]
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN role_name [, ...]
    | USER role_name [, ...]
    | SYSID uid
    | DEFAULT TABLESPACE tablespace_name
    | PROFILE DEFAULT
    | PROFILE profile_name
    | PGUSER
```

Parameters

- **user_name**

  Name of the user to be created.

  Value range: a string. It must comply with the **naming convention**. A value can contain a maximum of 63 characters.

> **CAUTION**
>
> If a username contains uppercase letters, the database automatically converts the uppercase letters into lowercase letters. To create a username that contains uppercase letters, enclose the username with double quotation marks ("").

- **password**

  Specifies the login password.

  The new password must:

  - Contain at least eight characters. This is the default length.
  - Differ from the username or the username spelled backward.
  - Contain at least three of the following character types: uppercase characters, lowercase characters, digits, and special characters (limited to ~!@#$%^&*()-_=+\|[{}];:,<.>/?).
  - The password can also be a ciphertext character string that meets the format requirements. This mode is mainly used to import user data. You are advised not to use it directly. If a ciphertext password is used, the user must know the plaintext corresponding to the ciphertext password and ensure that the password meets the complexity requirements. The database does not verify the complexity of the ciphertext password. Instead, the security of the ciphertext password is ensured by the user.
  - Be enclosed by single quotation marks when a user is created.

  Value range: a string

For other parameters, see **CREATE ROLE**.

Examples

```
-- Create user jim whose login password is ********.
gaussdb=# CREATE USER jim PASSWORD '********';

-- Create user kim whose login password is ********.
gaussdb=# CREATE USER kim IDENTIFIED BY '********';

-- Create user tom whose login password is ********.
gaussdb=# CREATE USER TOM PASSWORD '**********';

-- Create user TOM whose login password is ********.
gaussdb=# CREATE USER "TOM" PASSWORD '**********';

-- To create a user with the CREATEDB permission, add the CREATEDB keyword.
gaussdb=# CREATE USER dim CREATEDB PASSWORD '********';

-- Query the permissions of the dim user.
gaussdb=#  \du dim
          List of roles
 Role name | Attributes | Member of
-----------+------------+-----------
 dim       | Create DB  | {}
(You can see that the dim user has the CREATEDB permission.)

-- Change the login password of user jim.
gaussdb=# ALTER USER jim IDENTIFIED BY '**********' REPLACE '********';

-- Add the CREATEROLE permission to jim.
gaussdb=# ALTER USER jim CREATEROLE;
```



```
-- View the CREATEROLE permission added to user jim.
gaussdb=# \du jim
          List of roles
 Role name | Attributes  | Member of
-----------+-------------+-----------
 jim       | Create role | {}

-- Set enable_seqscan to on. (The setting will take effect in the next session.)
gaussdb=# ALTER USER jim SET enable_seqscan TO on;

-- Reset the enable_seqscan parameter for jim.
gaussdb=# ALTER USER jim RESET enable_seqscan;

-- Lock jim.
gaussdb=# ALTER USER jim ACCOUNT LOCK;

-- Unlock jim.
gaussdb=# ALTER USER jim ACCOUNT UNLOCK;

-- Change the user password.
gaussdb=# ALTER USER dim WITH PASSWORD '********';

-- Change the username.
gaussdb=# ALTER USER dim RENAME TO lisa;

-- Delete the user.
gaussdb=# DROP USER kim CASCADE;
gaussdb=# DROP USER jim CASCADE;
gaussdb=# DROP USER lisa CASCADE;
gaussdb=# DROP USER TOM CASCADE;
gaussdb=# DROP USER "TOM" CASCADE;
```

Helpful Links

ALTER USER, **CREATE ROLE**, and **DROP USER**

7.12.8.53 CREATE USER MAPPING

Description

CREATE USER MAPPING defines a new mapping from a user to an external server. A user mapping usually contains connection information. The external data wrapper uses the connection information and the information contained in the external server to access an external data source, the owner of an external server can create a user mapping for the server for any user. If a user is granted the USAGE privilege on the server, the user can create a user mapping for its own username.

Precautions

- If the **password** option is displayed, ensure that the **usermapping.key.cipher** and **usermapping.key.rand** files exist in the *$GAUSSHOME*/**bin** directory of each node in GaussDB. If the two files do not exist, use the gs_guc tool to generate them and use the gs_ssh tool to release them to the *$GAUSSHOME*/**bin** directory on each node in GaussDB. For details, see the description in **OPTIONS ({ option_name 'value' } [, ...])**.
- When multi-layer quotation marks are used for sensitive columns (such as **password**) in **OPTIONS**, the semantics is different from that in the scenario where quotation marks are not used. Therefore, sensitive columns are not identified for anonymization.

Syntax

```
CREATE USER MAPPING FOR { user_name | USER | CURRENT_USER | PUBLIC }
    SERVER server_name
    [ OPTIONS ( option 'value' [ , ... ] ) ];
```

Parameters

- **user_name**

  Specifies the name of an existing user to map to a foreign server.

  CURRENT_USER and **USER** match the name of the current user. When **PUBLIC** is specified, a public mapping is created and used when no mapping for a particular user is available.

- **server_name**

  Specifies the name of the existing server for a created user mapping.

- **OPTIONS ({ option_name ' value ' } [, ...])**

  Specifies options for user mapping. These options typically define the actual username and password of this mapping. The option name must be unique. The allowed option names and values are related to the foreign data wrapper of the server.

  NOTE

  - User passwords are encrypted and stored in the system catalog PG_USER_MAPPING. During the encryption, **usermapping.key.cipher** and **usermapping.key.rand** are used as the encryption password file and encryption factor. Before using the tool for the first time, create the two files, save the files to the *$GAUSSHOME*/**bin** directory on each node, and ensure that you have the read permission on the files. **gs_ssh** helps you quickly place files in the specified directory of each node.
    ```
    gs_ssh -c "gs_guc generate -o usermapping -S default -D $GAUSSHOME/bin"
    ```
  - If the **-S** parameter is set to default, a password is randomly generated. You can also specify a password for the **-S** parameter to ensure the security and uniqueness of the generated password file. You do not need to save or memorize the password. For details about other parameters, see the description of the **gs_guc** tool in the "Tool Reference".

Examples

```
-- Create a role.
gaussdb=# CREATE ROLE bob PASSWORD '********';

-- Create a foreign server.
gaussdb=# CREATE SERVER my_server FOREIGN DATA WRAPPER log_fdw;

-- Create a user mapping.
gaussdb=# CREATE USER MAPPING FOR bob SERVER my_server OPTIONS (USER 'bob', PASSWORD
'********');

-- Modify the user mapping.
gaussdb=# ALTER USER MAPPING FOR bob SERVER my_server OPTIONS (SET PASSWORD '********');

-- Delete the user mapping.
gaussdb=# DROP USER MAPPING FOR bob SERVER my_server;

-- Delete the foreign server.
gaussdb=# DROP SERVER my_server;

-- Delete the role.
gaussdb=# DROP ROLE bob;
```


Helpful Links

ALTER USER MAPPING and **DROP USER MAPPING**

7.12.8.54 CREATE VIEW

Description

Creates a view. A view is a virtual table. Only view definition is stored in the database and view data is not. The data is stored in a base table. If data in the base table changes, the data in the view changes accordingly. In this sense, a view is like a window through which users can know their interested data and data changes in the database.

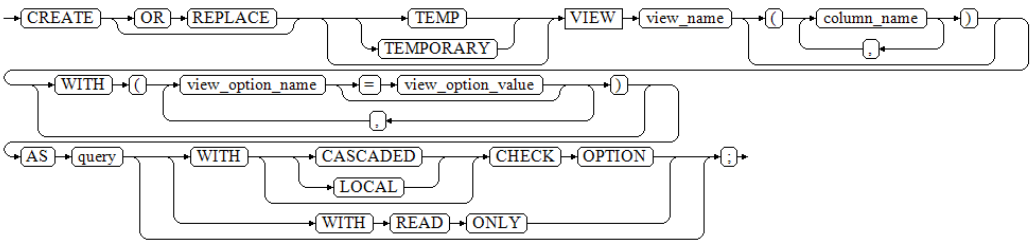
- You can define frequently used data as views to encapsulate complex query SQL statements, which simplifies operations.
- You can query only data defined in a view, which is secure. Columns in a base table are hidden to protect the data structure of the database.
- User permission management is simplified. Only the permission to use views is granted to users.

Precautions

A user granted with the CREATE ANY TABLE permission can create views in the public and user schemas.

Syntax

```
CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] VIEW view_name [ ( column_name [, ...] ) ]
    [ WITH ( {view_option_name [= view_option_value]} [, ... ] ) ]
    AS query
    [ WITH [ CASCADED | LOCAL ] CHECK OPTION  | WITH READ ONLY ];
```



NOTE

You can use WITH(security_barrier) to create a relatively secure view. This prevents attackers from printing hidden base table data by using the RAISE statement of low-cost functions.

After a view is created, you are not allowed to use REPLACE to modify column names in the view or delete the columns.

Parameters

- **OR REPLACE**

  (Optional) Redefines the view if it already exists.

- **TEMP | TEMPORARY**

(Optional) Creates a temporary view. The view is automatically deleted when the current session ends. If any table referenced by a view is a temporary table, the view is created as a temporary view (regardless of whether **TEMP|TEMPORARY** is specified in the SQL statement).

- **view_name**

  Specifies the name (optionally schema-qualified) of the view to be created.

  Value range: a string. It must comply with the **naming convention**.

- **column_name**

  Specifies an optional list of names to be used for columns of the view. If not given, the column names are deduced from the query.

  Value range: a string. It must comply with the **naming convention**.

- **view_option_name [= view_option_value]**

  Specifies an optional parameter for a view.

  Currently, view_option_name supports only **security_barrier** and **check_option**.

  security_barrier: This parameter is used when the view attempts to provide row-level security. Value range: Boolean (**true** or **false**).

  check_option: controls the behavior of updating a view. Value range: **CASCADED** and **LOCAL**.

- **query**

  Specifies a SELECT or VALUES statement that will provide the columns and rows of the view.

  NOTICE

  If **query** contains a clause specifying the partition of a partitioned table, the OID of the specified partition is fixed to the system catalog when the view is created. If a DDL operation that will change the OID of the specified partition is used, for example, DROP, SPLIT, or MERGE, the view is unavailable. In this case, you need to create a view.

- **WITH [CASCADED | LOCAL] CHECK OPTION**

  Controls the behavior of updatable views. INSERT and UPDATE on the view will be checked to ensure that new rows meet the view-defining condition, that is, the new rows are visible through the view. If the check fails, the modification is rejected. If this option is not added, INSERT and UPDATE on the view are allowed to create rows that are not visible through the view. **WITH CHECK OPTION** can be set to **CASCADED** or **LOCAL**:

  CASCADED: New rows are checked against the conditions of the view and all underlying views. If **CHECK OPTION** is specified, and neither **LOCAL** nor **CASCADED** is specified, then **CASCADED** is used by default.

  LOCAL: New rows are only checked against the conditions defined directly on the view itself. Any conditions defined on underlying views are not checked (unless they also specify **CHECK OPTION**).

- **WITH READ ONLY**

  A read-only view is defined. You cannot insert, update, or delete data in the view.

NOTE

- The concepts involved in the constraints on updating, inserting, and deleting views are described as follows:
  - Join view: view created with multiple tables using JOIN.
  - Key-preserved table: Insertion, update, and deletion of multi-table join views are restricted by key-preserved tables. In a multi-table view, if each row in the source table corresponds to each row in the view, and a row in the source table does not correspond to multiple rows in the view after the JOIN, the source table is a key-preserved table.
  - Relationship between the top layer and the bottom layer: A view may be nested at multiple layers. For example, a view consists of one or more views or subqueries. The view that is directly operated by the DML is called the top layer. The tables and views that form the view and the tables and views in the WITH clause are called the corresponding bottom layer.
  - Columns that can be updated: Columns that are not system columns or whole-row references and directly reference user columns in the base table can be updated. You can query the **is_updatable** column in **information_schema.columns** to check whether a column in a view or table can be updated.
  - Updatable view: view that can be inserted, updated, or deleted. Updatable views do not contain DISTINCT, GROUP BY, HAVING, LIMIT and OFFSET clauses, set operations (UNION, INTERSECT, and EXCEPT), aggregate functions, window functions, and return set functions (array_agg, json_agg, and generate_series). **WITH CHECK OPTION** can be defined only on updatable views. To check whether a view can be updated, query the **is_updatable** and **is_insertable_into** columns in **information_schema.views** or the **information_schema.tables.is_insertable_into** column. In **information_schema.views**, **is_updatable** specifies whether a view can be updated or deleted, **is_insertable_into** specifies whether data can be inserted into a view, and **information_schema.tables.is_insertable_into** specifies whether data can be inserted into a relationship.
- If an updatable view has an INSTEAD OF trigger or INSTEAD rule, using CHECK OPTION does not check the conditions on that view.
- If an underlying view of an updatable view has an INSTEAD OF trigger and the updatable view defines the **CHECK OPTION** option of the CASCADED type, in non-A-compatible mode, the conditions of the underlying view with the INSTEAD OF trigger are recursively checked, and the conditions of the underlying view with the trigger are not checked. In A-compatible mode, the **CHECK OPTION** option of all views is invalid and no check is performed.
- If **CASCADED CHECK OPTION** is defined in the upper-layer view and **LOCAL CHECK OPTION** is defined in the lower-layer view, **LOCAL CHECK OPTION** in the lower-layer view is overwritten by **CASCADED CHECK OPTION** in the upper-layer view.
- If an updatable view or any of its underlying relationships has an INSTEAD rule that causes INSERT or UPDATE rewriting, using CHECK OPTION will not check the conditions on all views.
- If **CHECK OPTION** is specified, insert and update operations cannot be performed on join columns in multi-table join views or multi-table join subqueries.
- If **CHECK OPTION** is specified, duplicate base tables exist in a multi-table join view or multi-table join subquery, and the duplicate base tables are not all key-preserved tables, the view or subquery cannot be deleted.

Examples

- Common view:

```
-- Create the test_tb1 table and insert 100 data records into the table.
gaussdb=# CREATE TABLE test_tb1(col1 int, col2 int);
```

```
gaussdb=# INSERT INTO test_tb1 VALUES (generate_series(1,100),generate_series(1,100));
-- Create a view whose value of col1 is less than 3.
gaussdb=# CREATE VIEW test_v1 AS SELECT * FROM test_tb1 WHERE col1 < 3;
-- Query a view.
gaussdb=# SELECT * FROM test_v1;
 col1 | col2
------+------
    1 |    1
    2 |    2
(2 rows)

-- Delete the table and the view.
gaussdb=# DROP VIEW test_v1;
gaussdb=# DROP TABLE test_tb1;
```

- Temporary view:

```
-- Create a table and a temporary view.
gaussdb=# CREATE TABLE test_tb2(c1 int, c2 int);
gaussdb=# CREATE TEMP VIEW test_v2 AS SELECT * FROM test_tb2;
-- Query the table and view information. (The temporary table belongs to a schema starting with
pg_temp instead of public.)
gaussdb=# \d
                                    List of relations
         Schema          |   Name   | Type  | Owner |                      Storage
-------------------------+----------+-------+-------+-----------------------------------------------------
 pg_temp_dn_6001_1_1_9473 | test_v2  | view  | omm   |
 public                   | test_tb2 | table | omm   |
{orientation=row,compression=no,storage_type=USTORE}
(2 rows)

-- Exit the current session and log in again. Check whether the temporary view is deleted.
gaussdb=# \d
                              List of relations
 Schema |   Name   | Type  | Owner |                      Storage
--------+----------+-------+-------+-----------------------------------------------------
 public | test_tb2 | table | omm   | {orientation=row,compression=no,storage_type=USTORE}
(1 row)

-- Delete the table.
gaussdb=# DROP TABLE test_tb2;
```

- Insert, update, and delete views.

```
-- Create a single table view.
gaussdb=# CREATE TABLE t_view_iud1 (x int, y int);
CREATE TABLE
gaussdb=# INSERT INTO t_view_iud1 VALUES (11, 11);
INSERT 0 1
gaussdb=# CREATE VIEW vt AS SELECT * FROM t_view_iud1;
CREATE VIEW
gaussdb=# CREATE VIEW vt_wco AS SELECT * FROM t_view_iud1 WHERE x > 5 WITH CHECK OPTION;
CREATE VIEW

-- View the information_schema.columns view and check whether columns in the view can be
updated.
gaussdb=# SELECT table_schema, table_name, column_name, is_updatable FROM
information_schema.columns WHERE table_schema = current_schema AND table_name = 'vt';
 table_schema | table_name | column_name | is_updatable
--------------+------------+-------------+--------------
 public       | vt         | y           | YES
 public       | vt         | x           | YES
(2 rows)

-- Insert and update data in a view. If WITH CHECK OPTION is specified when a view is created, data
is checked before the view is updated.
gaussdb=# INSERT INTO vt VALUES (1, 1);
INSERT 0 1
gaussdb=# INSERT INTO vt_wco VALUES (1, 1);
ERROR:  new row violates WITH CHECK OPTION for view "vt_wco"
DETAIL:  Failing row contains (1, 1).
gaussdb=# UPDATE vt SET y = 121 WHERE y = 1;
```



```
UPDATE 1
gaussdb=# UPDATE vt_wco SET y = 6 WHERE y = 11;
UPDATE 1
gaussdb=# DELETE FROM vt WHERE y =11;
DELETE 0

-- Create a multi-layer nested view and specify WITH CHECK OPTION in the bottom-layer relationship. Check the corresponding conditions when DML operations are performed on the top-layer view.
gaussdb=# CREATE VIEW vvtt AS SELECT * FROM vt_wco;
CREATE VIEW
gaussdb=# INSERT INTO vvtt VALUES (1, 1), (2, 2);
ERROR:  new row violates WITH CHECK OPTION for view "vt_wco"
DETAIL:  Failing row contains (1, 1).

-- View information_schema.tables and information_schema.views and check whether they can be inserted and updated.
gaussdb=# SELECT table_schema, table_name, is_insertable_into FROM information_schema.tables
gaussdb-# WHERE table_schema = current_schema AND table_name = 'vvtt';
 table_schema | table_name | is_insertable_into
--------------+------------+--------------------
 public       | vvtt       | YES
(1 row)

gaussdb=# SELECT table_name, is_updatable, check_option FROM information_schema.views
gaussdb-# WHERE table_schema = current_schema AND table_name = 'vvtt';
 table_name | is_updatable | check_option
------------+--------------+--------------
 vvtt       | YES          | NONE
(1 row)

gaussdb=# SELECT table_name, is_updatable, check_option FROM information_schema.views WHERE table_schema = current_schema AND table_name = 'vt_wco';
 table_name | is_updatable | check_option
------------+--------------+--------------
 vt_wco     | YES          | CASCADED
(1 row)

-- Delete the view and table.
gaussdb=# DROP VIEW vvtt, vt, vt_wco CASCADE;
DROP VIEW
gaussdb=# DROP TABLE t_view_iud1;
DROP TABLE
```

Helpful Links

ALTER VIEW and **DROP VIEW**

7.12.8.55 CREATE WEAK PASSWORD DICTIONARY

Description

CREATE WEAK PASSWORD DICTIONARY creates a weak password dictionary, which is empty by default. You can use this syntax to add one or more weak passwords to the gs_global_config system catalog.

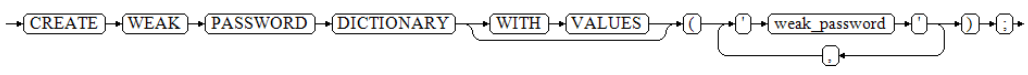
Precautions

- Only the initial user, system administrator, and security administrator have the permission to execute this syntax.
- Passwords in the weak password dictionary are stored in the gs_global_config system catalog.

- The weak password dictionary is empty by default. You can use this syntax to add one or more weak passwords.
- When a user attempts to execute this syntax to insert a weak password that already exists in the gs_global_config system catalog, only one weak password is retained in the system catalog.

Syntax

```
CREATE WEAK PASSWORD DICTIONARY
    [WITH VALUES] ( {'weak_password'} [, ...] );
```



Parameters

- **weak_password**

  Specifies a weak password.

  Value range: a string.

Examples

```
-- Insert a single weak password into the gs_global_config system catalog.
gaussdb=# CREATE WEAK PASSWORD DICTIONARY WITH VALUES ('********');

-- Check weak passwords in the gs_global_config system catalog.
gaussdb=# SELECT * FROM gs_global_config WHERE NAME LIKE 'weak_password';
     name       |    value
----------------+--------------
 weak_password   | ********
(1 rows)

-- Insert multiple weak passwords into the gs_global_config system catalog.
gaussdb=# CREATE WEAK PASSWORD DICTIONARY WITH VALUES ('********'),('********');

-- Check the weak password in the gs_global_config system catalog again. The asterisk (*) in a weak
password does not represent the password content.
gaussdb=# SELECT * FROM gs_global_config WHERE NAME LIKE 'weak_password';
     name       |     value
----------------+----------------
 weak_password   | ********
 weak_password   | ********
 weak_password   | ********
(3 rows)

-- Clear all weak passwords in the gs_global_config system catalog.
gaussdb=# DROP WEAK PASSWORD DICTIONARY;

-- View existing weak passwords.
gaussdb=# SELECT * FROM gs_global_config WHERE NAME LIKE 'weak_password';
 name | value
------+-------
(0 rows)
```

Helpful Links

DROP WEAK PASSWORD DICTIONARY

7.12.8.56 CURSOR

Description

This statement is used to create a cursor and retrieve specified rows of data from a query.

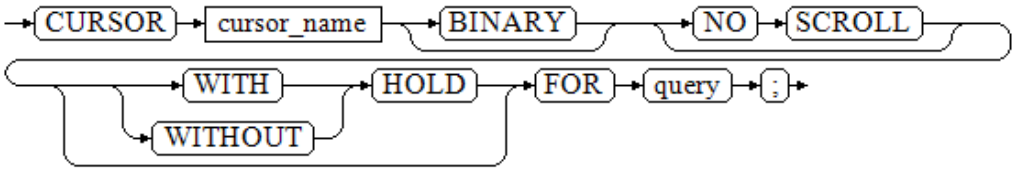
To process SQL statements, the stored procedure thread assigns a memory segment to store context association. Cursors are handles or pointers pointing to context regions. With cursors, stored procedures can control alterations in context regions.

Precautions

- CURSOR is used only in transaction blocks.
- Generally, CURSOR and SELECT both have text returns. Since data is stored in binary format in the system, the system needs to convert the data from the binary format to the text format. If data is returned in text format, client applications need to convert the data back to the binary format for processing. FETCH implements conversion between binary data and text data.
- Binary cursors should be used carefully. Text usually occupies larger space than binary data. A binary cursor returns internal binary data, which is easier to operate. A text cursor returns text, which is easier to retrieve and therefore reduces workload on the client. As an example, if a query returns a value of one from an integer column, you would get a string of 1 with a default cursor, whereas with a binary cursor you would get a 4-byte field containing the internal representation of the value (in big-endian byte order).
- Parameters can be defined for static cursors. The parameters support default values. However, the default value of a parameter cannot be a variable outside the package. You can define a variable in the package, assign the variable outside the package to the defined variable, and use the variable as the default value.
- The definition of a cursor generates a query object. Currently, stored procedures do not support the creation of dependencies on query objects.

Syntax

```
CURSOR cursor_name
    [ BINARY ]  [ NO SCROLL ]  [ { WITH | WITHOUT } HOLD ]
    FOR query;
```



Parameters

- **cursor_name**

  Specifies the name of the cursor to be created.

  Value range: a string. It must comply with the naming convention.

- **BINARY**

  Specifies that data retrieved by a cursor will be returned in binary format, not in text format.

- **NO SCROLL**

  Specifies how the cursor retrieves rows.

  – **NO SCROLL**: specifies that the cursor cannot be used to retrieve rows in a nonsequential fashion.

  – Unspecified: Based on the query's execution plan, the system automatically determines whether the cursor can be used to retrieve rows in a nonsequential fashion.

  NOTE

  In cmdsql, only the NO SCROLL keyword can be added or the scrolling option is not added. The SCROLL keyword cannot be added.

  In a stored procedure, the keywords NO SCROLL and SCROLL can be added, or the scrolling option is not added.

  Among explicit cursors, only cursors declared as **NO SCROLL** can be concurrently executed. You are advised to set this parameter to **NO SCROLL** for cursors that do not need to retrieve data rows in reverse order.

- **WITH HOLD | WITHOUT HOLD**

  Specifies whether a cursor can be used after the transaction that created it ends.

  – **WITH HOLD**: The cursor can be used after the transaction that created it ends.

  – **WITHOUT HOLD**: The cursor cannot be used outside of the transaction that created it.

  – If neither **WITH HOLD** nor **WITHOUT HOLD** is specified, the default is **WITHOUT HOLD**.

- **query**

  Uses the **SELECT** or **VALUES** clause to specify the rows to be returned by a cursor.

  Value range: **SELECT** or **VALUES** clause

Examples

Example 1:

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE tbl_test(c1 int);
gaussdb=# INSERT INTO tbl_test VALUES (generate_series(1,20));

-- Set up cursor1.
gaussdb=# BEGIN;
gaussdb=# CURSOR cursor1 FOR SELECT * FROM tbl_test ORDER BY 1;

-- Run the FETCH command to retrieve three rows of data.
gaussdb=# FETCH FORWARD 3 FROM cursor1;
 c1
----
  1
  2
  3
(3 rows)
```



```
-- Run the MOVE command to move the cursor backwards by two rows. No result is returned.
gaussdb=# MOVE FORWARD 2 FROM cursor1;
MOVE 2

-- Run the FETCH command to retrieve two rows of data.
gaussdb=# FETCH FORWARD 2 FROM cursor1;
 c1
----
  6
  7
(2 rows)

-- Close the cursor and end the transaction.
gaussdb=# CLOSE cursor1;
gaussdb=# END;
```

Example 2: Cursor with the WITH HOLD attribute

```
-- Set up a WITH HOLD cursor named cursor2.
gaussdb=# BEGIN;
gaussdb=# CURSOR cursor2 WITH HOLD FOR SELECT * FROM tbl_test ORDER BY 1;

-- Run the FETCH command to retrieve three rows of data.
gaussdb=# FETCH FORWARD 3 FROM cursor2;
 c1
----
  1
  2
  3
(3 rows)

-- Different from common cursors, cursors with the WITH HOLD attribute can still be used after a
transaction ends.
gaussdb=# END;
gaussdb=# FETCH FORWARD 2 FROM cursor2;
 c1
----
  4
  5
(2 rows)

-- Close the cursor.
gaussdb=# CLOSE cursor2;

-- Delete.
gaussdb=# DROP TABLE tbl_test;
```

Helpful Links

8.12.11.1-FETCH, CLOSE, and MOVE

7.12.9 D

7.12.9.1 DEALLOCATE

Description

DEALLOCATE deallocates prepared statements.

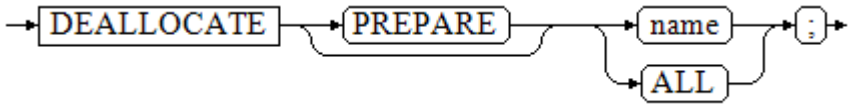
Precautions

- If you do not explicitly deallocate a prepared statement, it is deallocated when the session ends.

- The PREPARE keyword in the syntax is always ignored.

Syntax

```
DEALLOCATE [ PREPARE ] { name | ALL };
```



Parameters

- **name**

  Specifies the name of the prepared statement to be deallocated.

- **ALL**

  Deallocates all prepared statements.

Examples

```
-- View existing prepared statements.
gaussdb=# SELECT name, statement, parameter_types FROM pg_prepared_statements;
 name | statement | parameter_types
------+-----------+-----------------
(0 rows)

-- Create four prepared statements q1, q2, q3, and q4.
gaussdb=# PREPARE q1 AS SELECT 1 AS a;
PREPARE
gaussdb=# PREPARE q2 AS SELECT 1 AS a;
PREPARE
gaussdb=# PREPARE q3 AS SELECT 1 AS a;
PREPARE
gaussdb=# PREPARE q4 AS SELECT 1 AS a;
PREPARE

-- View existing prepared statements again.
gaussdb=# SELECT name, statement, parameter_types FROM pg_prepared_statements;
 name |          statement          | parameter_types
------+-----------------------------+-----------------
 q1   | PREPARE q1 AS SELECT 1 AS a; | {}
 q4   | PREPARE q4 AS SELECT 1 AS a; | {}
 q3   | PREPARE q3 AS SELECT 1 AS a; | {}
 q2   | PREPARE q2 AS SELECT 1 AS a; | {}
(4 rows)

-- Delete the prepared statements q4 and view the remaining prepared statements.
gaussdb=# DEALLOCATE q4;
gaussdb=# SELECT name, statement, parameter_types FROM pg_prepared_statements;
 name |          statement          | parameter_types
------+-----------------------------+-----------------
 q1   | PREPARE q1 AS SELECT 1 AS a; | {}
 q3   | PREPARE q3 AS SELECT 1 AS a; | {}
 q2   | PREPARE q2 AS SELECT 1 AS a; | {}
(3 rows)

-- Delete all prepared statements and view the remaining prepared statements.
gaussdb=# DEALLOCATE ALL;
DEALLOCATE ALL
gaussdb=# SELECT name, statement, parameter_types FROM pg_prepared_statements;
 name | statement | parameter_types
------+-----------+-----------------
(0 rows)
```


7.12.9.2 DECLARE

Description

DECLARE defines a cursor to retrieve a small number of rows at a time out of a larger query and can be the start of an anonymous block.

This section describes usage of cursors. The usage of anonymous blocks is available in **BEGIN**.

To process SQL statements, the stored procedure thread assigns a memory segment to store context association. Cursors are handles or pointers pointing to context regions. With cursors, stored procedures can control alterations in context regions.

Generally, CURSOR and SELECT both have text returns. Since data is stored in binary format in the system, the system needs to convert the data from the binary format to the text format. If data is returned in text format, client applications need to convert the data back to the binary format for processing. FETCH implements conversion between binary data and text data.

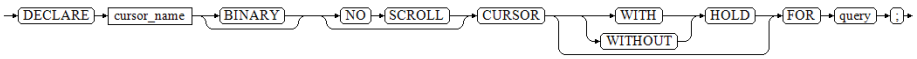
Precautions

- CURSOR is used only in transaction blocks.
- Binary cursors should be used carefully. Text usually occupies larger space than binary data. A binary cursor returns internal binary data, which is easier to operate. A text cursor returns text, which is easier to retrieve and therefore reduces workload on the client. As an example, if a query returns a value of one from an integer column, you would get a string of 1 with a default cursor, whereas with a binary cursor you would get a 4-byte value containing the internal representation of the value (in big-endian byte order).

Syntax

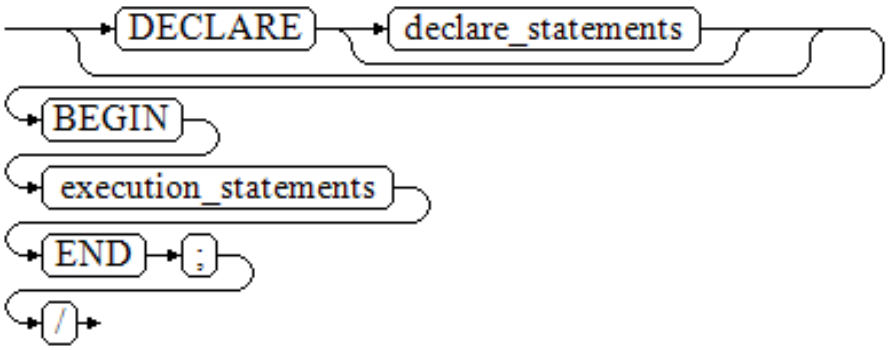
- Define a cursor.

```
DECLARE cursor_name [ BINARY ] [ NO SCROLL ]
    CURSOR [ { WITH | WITHOUT } HOLD ] FOR query;
```



- Enable an anonymous block.

```
[DECLARE [declare_statements]]
BEGIN
execution_statements
END;
/
```



Parameters

- **cursor_name**

  Specifies the name of the cursor to be created.

  Value range: a string. It must comply with the naming convention.

- **BINARY**

  Causes the cursor to return data in binary rather than in text format.

- **NO SCROLL**

  Specifies how the cursor retrieves rows.

  - **NO SCROLL**: specifies that the cursor cannot be used to retrieve rows in a nonsequential fashion.
  - Unspecified: Based on the query's execution plan, the system automatically determines whether the cursor can be used to retrieve rows in a nonsequential fashion.

- **WITH HOLD**

  WITHOUT HOLD

  Specifies whether the cursor can continue to be used after the transaction that created it successfully commits.

  - **WITH HOLD**: The cursor can continue to be used after the transaction that created it successfully commits.
  - **WITHOUT HOLD**: The cursor cannot be used outside of the transaction that created it.
  - If neither **WITH HOLD** nor **WITHOUT HOLD** is specified, the default is **WITHOUT HOLD**.

  NOTICE

  - For a cursor declared as **WITH HOLD**, all data of the cursor is cached when a transaction ends. If the cursor has a large amount of data, this process may take a long time.

- **query**

  Uses a **SELECT** or **VALUES** clause to specify the rows to be returned by the cursor.

  Value range: **SELECT** or **VALUES** clause

- **declare_statements**

  Declares a variable, including its name and type, for example, **sales_cnt int**.

- **execution_statements**

  Specifies the statement to be executed in an anonymous block.

  Value range: an existing function name

Examples

For details about how to define a cursor, see **Examples** in section "FETCH."

Helpful Links

BEGIN and **FETCH**

7.12.9.3 DELETE

Description

Deletes rows that satisfy the WHERE clause from the specified table. If the WHERE clause is absent, the effect is to delete all rows in the table. The result is a valid, but an empty table.

Precautions

- The owner of a table, users granted with the DELETE permission on the table, or users granted with the DELETE ANY TABLE permission can delete data from the table. When separation of duties is disabled, a system administrator has the permission to delete data from the table by default, as well as the SELECT permission on any table in the USING clause or whose values are read in condition.
- For row-store replication tables, DELETE can be performed only in the following two scenarios:
  - Scenarios with primary key constraints.
  - Scenarios where the execution plan can be pushed down.
- The syntax for deleting multiple tables is not applicable to views and tables containing RULE.
- For a DELETE statement whose subquery is a STREAM plan, UPDATE cannot be performed on the deleted row data.

Syntax

Delete a single table:

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
DELETE [/*+ plan_hint */] [FROM] [ ONLY ] {table_name [ * ] [ [ [partition_clause]  [ [ AS ] alias ] ] |
[ [ [ AS ] alias ] [partitions_clause] ] ] | subquery [ [ AS ] alias ] | view_name [ [ AS ] alias ]}
    [ USING using_list ]
    [ WHERE condition | WHERE CURRENT OF cursor_name ]
    [ ORDER BY { expression  [ ASC | DESC | USING operator ] } ]
    [ LIMIT { count } ]
    [ RETURNING { * | { output_expr [ [ AS ] output_name ] } [, ...] } ];
```

Delete multiple tables:

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
DELETE [/*+ plan_hint */] [FROM]
    {[ ONLY ] table_name [ * ] [ [ [partition_clause]  [ [ AS ] alias ] ] | [ [ [ AS ] alias ]
[partitions_clause] ] ]} [, ...]
    [ USING using_list ]
    [ WHERE condition ];
```

or

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
DELETE [/*+ plan_hint */]
    {[ ONLY ] table_name [ * ] [ [ [partition_clause]  [ [ AS ] alias ] ] | [ [ [ AS ] alias ]
[partitions_clause] ] ]} [, ...]
```



```
  [ FROM using_list ]
  [ WHERE condition ];
```

Format of **with_query**:

```
with_query_name [ ( column_name [, ...] ) ] AS [ [ NOT ] MATERIALIZED ]
( {select | values | insert | update | delete} )
```

Parameters

- **WITH [RECURSIVE] with_query [, ...]**

  Specifies one or more subqueries that can be referenced by name in the main query, which is equivalent to a temporary table.

  If RECURSIVE is specified, it allows a SELECT subquery to reference itself by name.

  - **with_query_name** specifies the name of the result set generated by a subquery. Such names can be used to access the result sets of subqueries in a query.

    column_name specifies the column name displayed in the subquery result set.
  - Each subquery can be a SELECT, VALUES, INSERT, UPDATE, or DELETE statement.
  - You can use MATERIALIZED or NOT MATERIALIZED to modify the CTE.
    - If **MATERIALIZED** is specified, the WITH query will be materialized, and a copy of the subquery result set is generated. The copy is directly queried at the reference point. Therefore, the WITH subquery cannot be jointly optimized with the SELECT statement trunk (for example, predicate pushdown and equivalence class transfer). In this scenario, you can use **NOT MATERIALIZED** for modification. If the WITH query can be executed as a subquery inline, the preceding optimization can be performed.
    - If the user does not explicitly declare the materialized attribute, comply with the following rules: If the CTE is referenced only once in the trunk statement to which it belongs and semantically supports inline execution, it will be rewritten as subquery inline execution. Otherwise, the materialized execution will be performed in CTE Scan mode.

- **plan_hint**

  Follows the DELETE keyword in the **/*+ */** format. It is used to optimize the plan of a DELETE statement block. For details, see **Hint-based Tuning**. In each statement, only the first **/*+** *plan_hint* ***/** comment block takes effect as a hint. Multiple hints can be written.

- **ONLY**

  If **ONLY** is specified before the table name, matching rows are deleted from the named table only. If **ONLY** is not specified, matching rows are also deleted from any tables inheriting from the named table.

- **table_name**

  Specifies the name (optionally schema-qualified) of the target table.

  Value range: an existing table name

> **NOTE**
>
> You can use database links to perform operations on remote tables. For details, see **DATABASE LINK**.

- **subquery**

  The object to be deleted can be a subquery. When data in a subquery is deleted, the subquery is regarded as a temporary view. The **CHECK OPTION** option can be added to the end of the subquery.

  ```
  [ WITH [ RECURSIVE ] with_query [, ...] ]
  SELECT [/*+ plan_hint */] [ ALL ]
  { * | {expression [ [ AS ] output_name ]} [, ...] }
  [ into_option ]
  [ FROM from_item [, ...] ]
  [ WHERE condition ]
  [ [ START WITH condition ] CONNECT BY [NOCYCLE] condition [ ORDER SIBLINGS BY expression ] ]
  [ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS
  { FIRST | LAST } ]} [, ...] ]
  [ OFFSET start [ ROW | ROWS ] ]
  [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
  [ into_option ];
  ```

  The specified subquery source **from_item** is as follows:

  ```
  {[ ONLY ] {table_name | view_name} [ * ] [ partition_clause ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
  |( select ) [ AS ] alias [ ( column_alias [, ...] ) ]
  |with_query_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
  |from_item [ NATURAL ] join_type from_item [ ON join_condition | USING ( join_column [, ...] ) ]}
  ```

  If there is only one table in the subquery, the table is deleted. If there are multiple tables or nested tables in the subquery, the system determines whether the table can be deleted by checking whether there is a key-preserved table. For details about key-preserved tables and **WITH CHECK OPTION**, see **CREATE VIEW**.

- **view_name**

  Specifies name of the target view.

NOTE

The restrictions on deleting views and subqueries are as follows:

- The DELETE operation can be performed only on columns that directly reference user columns in the base table.
- A subquery or view must contain at least one updatable column. For details about updatable columns, see **CREATE VIEW**.
- Views and subqueries that contain the DISTINCT, GROUP BY, HAVING, LIMIT or OFFSET clause at the top layer are not supported.
- Views and subqueries that contain set operations (UNION, INTERSECT, EXCEPT, and MINUS) at the top layer are not supported.
- Views and subqueries whose target lists contain aggregate functions, window functions, or return set functions (such as array_agg, json_agg, and generate_series) are not supported.
- Views with BEFORE or AFTER triggers but without INSTEAD OF triggers or INSTEAD rules are not supported.
- Table types supported in views and subqueries include ordinary tables, temporary tables, global temporary tables, partitioned tables, level-2 partitioned tables, Ustore tables, and Astore tables.
- A join view or subquery can delete only the key-preserved tables in the view or subquery. If there is only one key-preserved table, the data in the table is deleted. If there are multiple key-preserved tables, only the data in the first key-preserved table following **from** is deleted.
- If the **CHECK OPTION** option is specified in the join view or subquery, the base table is duplicate, and the duplicate base table retains inconsistent key table attributes in the view or subquery, rows cannot be deleted from the join view or subquery. For details about the key-preserved table, see **CREATE VIEW**.
- The DELETE operation cannot be performed on the system view.
- Deleting multiple tables is not supported.

- **partition_clause**

  Deletes a specified partition.

  PARTITION { (partition_name) | FOR (partition_value [, ...]) } |

  SUBPARTITION { (subpartition_name) | FOR (subpartition_value [, ...]) }

  For details about the keywords, see **SELECT**.

  For details, see **CREATE TABLE SUBPARTITION**.

- **partitions_clause**

  Deletes multiple partitions.

  PARTITION { ({ partition_name | subpartition_name } [, ...]) }

  This syntax takes effect only when **sql_compatibility** is set to **B**.

  For details about the keywords, see **SELECT**.

  For details, see **CREATE TABLE SUBPARTITION**.

- **alias**

  Specifies a substitute name for the target table.

  Value range: a string. It must comply with the **naming convention**.

- **using_list**

  Specifies the USING clause.

NOTICE

When **sql_compatibility** is set to **B** or multiple tables are to be deleted, the target tables can appear at the same time when **using_list** specifies the set of associated tables. In addition, the aliases of the tables can be defined and used in the target tables. In other situations, the target tables cannot appear repeatedly in **using_list**.

- **condition**

  Specifies an expression that returns a Boolean value. Only rows for which this expression returns **true** will be deleted. You are advised not to use numeric types such as int as conditions, because such types can be implicitly converted to bool values (non-zero values are implicitly converted to **true** and **0** is implicitly converted to **false**), which may cause unexpected results.

- **WHERE CURRENT OF cursor_name**

  When the cursor points to a row in a table, you can use this syntax to delete the row. For details about the restrictions, see **UPDATE**.

- **ORDER BY**

  For details about the keywords, see **SELECT**.

- **LIMIT**

  For details about the keywords, see **SELECT**.

- **output_expr**

  Specifies an expression used to calculate the output result after a row is deleted using the DELETE statement. The expression can use any column of the table. You can use * to return all columns of the deleted row.

- **output_name**

  Specifies a name to use for a returned column.

  Value range: a string. It must comply with the **naming convention**.

Examples

- Deleting some data records

```
-- Create a table.
gaussdb=# CREATE TABLE test_t1(col1 INT,col2 INT);
gaussdb=# INSERT INTO test_t1 VALUES (1, 1), (2, 2), (3, 3), (4, 4), (4, 6);

-- Delete some data records from the table.
gaussdb=# DELETE FROM test_t1 WHERE col1 = 4;

-- Query.
gaussdb=# SELECT * FROM test_t1;
 col1 | col2
------+------
    1 |    1
    2 |    2
    3 |    3
(3 rows)
```

- Deleting all data

```
-- Delete all data.
gaussdb=# DELETE FROM test_t1;

-- Query.
gaussdb=# SELECT * FROM test_t1;
 col1 | col2
```



```
------+------
(0 rows)

-- Delete the table.
gaussdb=# DROP TABLE test_t1;
```

- **WITH [RECURSIVE] with_query [, ...]**

```
-- Student table.
gaussdb=# CREATE TABLE student(id INT,name varchar(50));

-- Grade table.
gaussdb=# CREATE TABLE grade(id INT,score CHAR);

gaussdb=# INSERT INTO student VALUES (1, 'tom'), (2, 'jerry'), (3, 'david');
gaussdb=# INSERT INTO grade VALUES (1, 'A'), (2, 'B'), (3, 'b');

-- Delete the data whose id is 2 from the student table and delete the data of the student from the grade table.
gaussdb=# WITH del_stu AS(DELETE FROM student WHERE id = 2 RETURNING id)
    DELETE FROM grade WHERE id = (SELECT id FROM del_stu);

-- Query data.
gaussdb=# SELECT * FROM student;
 id | name
----+-------
  1 | tom
  3 | david
(2 rows)

gaussdb=# SELECT * FROM grade;
 id | score
----+-------
  1 | A
  3 | b
(2 rows)

-- Delete the table.
gaussdb=# DROP TABLE grade;
gaussdb=# DROP TABLE student;
```

- Deleting a view or subquery

  Example 1: Deleting a subquery

```
-- Create a schema.
gaussdb=# CREATE SCHEMA del_subqry;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA = 'del_subqry';
SET

-- Create tables and insert data into the tables.
gaussdb=# CREATE TABLE t1 (x1 int, y1 int);
CREATE TABLE
gaussdb=# CREATE TABLE t2 (x2 int PRIMARY KEY, y2 int);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "t2_pkey" for table "t2"
CREATE TABLE
gaussdb=# CREATE TABLE tdata (x INT PRIMARY KEY, y INT);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "tdata_pkey" for table "tdata"
CREATE TABLE
gaussdb=# CREATE TABLE tinfo (z INT PRIMARY KEY, comm VARCHAR2(20));
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "tinfo_pkey" for table "tinfo"
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES (1, 1), (2, 2), (3, 3), (5, 5);
INSERT 0 4
gaussdb=# INSERT INTO t2 VALUES (1, 1), (2, 2), (3, 3), (5, 5);
INSERT 0 4
gaussdb=# INSERT INTO tdata VALUES (1, 1), (2, 2), (3, 3);
INSERT 0 3
gaussdb=# INSERT INTO tinfo VALUES (1,'one'), (2, 'two'), (3, 'three'), (5, 'wrong three');
INSERT 0 4
```



```
-- Delete data from t1 using a subquery.
gaussdb=# DELETE FROM (SELECT * FROM t1) where y1 = 3;
DELETE 1

-- The subquery is read-only, and data cannot be deleted.
gaussdb=# DELETE FROM (SELECT * FROM t1 WITH READ ONLY) WHERE y1 = 1;
ERROR:  cannot perform a DML operation on a read-only subquery.

-- Delete a subquery of a multi-table join.
gaussdb=# SELECT * FROM t1, t2 WHERE x1 = x2;
 x1 | y1 | x2 | y2
----+----+----+----
  1 |  1 |  1 |  1
  2 |  2 |  2 |  2
  5 |  5 |  5 |  5
(3 rows)
gaussdb=# DELETE FROM (SELECT * FROM t1, t2 WHERE x1 = x2) WHERE y2 = 5;
DELETE 1

gaussdb=# SELECT * FROM t1, t2 WHERE x1 = x2;
 x1 | y1 | x2 | y2
----+----+----+----
  1 |  1 |  1 |  1
  2 |  2 |  2 |  2
(2 rows)

-- The subquery contains CHECK OPTION, and the tdata table is duplicate. td1 is not a key-preserved table, and td2 is a key-preserved table.
gaussdb=# DELETE FROM (SELECT td1.x x1, td1.y y1, td2.x x2, td2.y y2 FROM tdata td1, tdata td2, tinfo WHERE td2.y=tinfo.z AND td1.x=td2.y WITH CHECK OPTION) WHERE y1 = 2;
ERROR:  cannot delete from view without exactly one key-preserved table
-- If CHECK OPTION is not specified, a subquery with the same structure is created and deleted successfully.
gaussdb=# DELETE FROM (SELECT td1.x x1, td1.y y1, td2.x x2, td2.y y2 FROM tdata td1, tdata td2, tinfo WHERE td2.y=tinfo.z AND td1.x=td2.y) WHERE y1 = 2;
DELETE 1

-- Delete a schema.
gaussdb=# RESET CURRENT_SCHEMA;
RESET
gaussdb=# DROP SCHEMA del_subqry CASCADE;
NOTICE:  drop cascades to 4 other objects
DETAIL:  drop cascades to table del_subqry.t1
drop cascades to table del_subqry.t2
drop cascades to table del_subqry.tdata
drop cascades to table del_subqry.tinfo
DROP SCHEMA
```

Example 2: Deleting a view

```
-- Create a schema.
gaussdb=# CREATE SCHEMA del_view;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA = 'del_view';
SET

-- Create tables and insert data into the tables.
gaussdb=# CREATE TABLE t1 (x1 int, y1 int);
CREATE TABLE
gaussdb=# CREATE TABLE t2 (x2 int PRIMARY KEY, y2 int);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "t2_pkey" for table "t2"
CREATE TABLE
gaussdb=# CREATE TABLE tdata (x INT PRIMARY KEY, y INT);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "tdata_pkey" for table "tdata"
CREATE TABLE
gaussdb=# CREATE TABLE tinfo (z INT PRIMARY KEY, comm VARCHAR2(20));
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "tinfo_pkey" for table "tinfo"
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES (1, 1), (2, 2), (3, 3), (5, 5);
INSERT 0 4
```



```
gaussdb=# INSERT INTO t2 VALUES (1, 1), (2, 2), (3, 3), (5, 5);
INSERT 0 4
gaussdb=# INSERT INTO tdata VALUES (1, 1), (2, 2), (3, 3);
INSERT 0 3
gaussdb=# INSERT INTO tinfo VALUES (1,'one'), (2, 'two'), (3, 'three'), (5, 'wrong three');
INSERT 0 4

-- Create a single table view.
gaussdb=# CREATE VIEW v_del1 AS SELECT * FROM t1;
CREATE VIEW
gaussdb=# CREATE VIEW v_del_read AS SELECT * FROM t1 WITH READ ONLY;
CREATE VIEW

-- Delete data from t1 using a view.
gaussdb=# DELETE FROM v_del1 where y1 = 3;
DELETE 1

-- The view is read-only, and data cannot be deleted.
gaussdb=# DELETE FROM v_del_read WHERE y1 = 1;
ERROR:  cannot perform a DML operation on a read-only subquery.

-- Create multi-table join views.
gaussdb=# CREATE VIEW vvt1t2 AS SELECT * FROM t1, t2 WHERE x1 = x2;
CREATE VIEW
gaussdb=# CREATE VIEW vv_dup AS SELECT td1.x x1, td1.y y1, td2.x x2, td2.y y2 FROM tdata td1,
tdata td2, tinfo WHERE td2.y=tinfo.z AND td1.x=td2.y;
CREATE VIEW
gaussdb=# CREATE VIEW vv_dup_wco AS SELECT td1.x x1, td1.y y1, td2.x x2, td2.y y2 FROM tdata td1,
tdata td2, tinfo WHERE td2.y=tinfo.z AND td1.x=td2.y WITH CHECK OPTION;
CREATE VIEW

-- Delete a multi-table join view.
gaussdb=# SELECT * FROM vvt1t2;
 x1 | y1 | x2 | y2
----+----+----+----
  1 |  1 |  1 |  1
  2 |  2 |  2 |  2
  5 |  5 |  5 |  5
(3 rows)

gaussdb=# DELETE FROM vvt1t2 WHERE y2 = 5;
DELETE 1

gaussdb=# SELECT * FROM vvt1t2;
 x1 | y1 | x2 | y2
----+----+----+----
  1 |  1 |  1 |  1
  2 |  2 |  2 |  2
 (2 rows)

-- The view contains CHECK OPTION, and the tdata table is duplicate. td1 is not a key-preserved
table, and td2 is a key-preserved table.
gaussdb=# DELETE FROM vv_dup_wco WHERE y1 = 2;
ERROR:  cannot delete from view without exactly one key-preserved table
-- If CHECK OPTION is not specified, a view with the same structure is created and deleted
successfully.
gaussdb=# DELETE FROM vv_dup WHERE y1 = 2;
DELTE 1

-- Delete a schema.
gaussdb=# RESET CURRENT_SCHEMA;
RESET
gaussdb=# DROP SCHEMA del_view CASCADE;
NOTICE:  drop cascades to 9 other objects
DETAIL:  drop cascades to table del_view.t1
drop cascades to table del_view.t2
drop cascades to table del_view.tdata
drop cascades to table del_view.tinfo
drop cascades to view del_view.v_del1
```



```
drop cascades to view del_view.v_del_read
drop cascades to view del_view.vvt1t2
drop cascades to view del_view.vv_dup
drop cascades to view del_view.vv_dup_wco
DROP SCHEMA
```

Suggestions

- DELETE

  To delete all records in a table, use the TRUNCATE syntax.

7.12.9.4 DO

Description

Executes an anonymous code block.

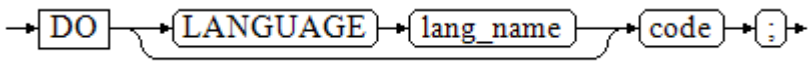
A code block is regarded as a function body without parameters. The return value type is void. It is parsed and executed a single time.

Precautions

- The procedural language to be used must already have been installed into the current database by means of CREATE LANGUAGE. PL/pgSQL is installed by default. To install another language, you must specify it.
- To use an untrusted language, you must have the USAGE permission on the programming language or the system administrator permission.

Syntax

```
DO [ LANGUAGE lang_name ] code;
```



Parameters

- **lang_name**

  Specifies the name of the procedural language the code is written in. If omitted, the default language is PL/pgSQL.

- **code**

  Specifies the programming language code that can be executed. The value must be a character string.

Examples

```
-- Create the webuser user.
gaussdb=# CREATE USER webuser PASSWORD '********';

-- Grant all permissions on all views in the tpcds schema to the webuser user.
gaussdb=# DO $$DECLARE r record;
BEGIN
    FOR r IN SELECT c.relname table_name,n.nspname table_schema FROM pg_class c,pg_namespace n
            WHERE c.relnamespace = n.oid AND n.nspname = 'tpcds' AND relkind IN ('r','v')
    LOOP
        EXECUTE 'GRANT ALL ON ' || quote_ident(r.table_schema) || '.' || quote_ident(r.table_name) || ' TO
webuser';
```



```
    END LOOP;
END$$;

-- Delete the webuser user.
gaussdb=# DROP USER webuser CASCADE;
```

7.12.9.5 DROP AGGREGATE

Description

Deletes an aggregate function.

Precautions

DROP AGGREGATE deletes an existing aggregate function. Only the owner of the aggregate function can run this command.

Syntax

```
DROP AGGREGATE [ IF EXISTS ] name ( argtype [ , ... ] ) [ CASCADE | RESTRICT ];
```

Parameters

- **IF EXISTS**

  Do not throw an error if the specified aggregation does not exist. A notice is issued in this case.

- **name**

  Existing aggregate function name (optionally schema-qualified).

- **argtype**

  Input data type of the aggregate function. To reference a zero-parameter aggregate function, use * to replace the input data type list.

- **CASCADE**

  Cascade deletes objects that depend on the aggregate function.

- **RESTRICT**

  Refuses to delete the aggregate function if any objects depend on it. This is a default processing.

Examples

```
-- Create a user-defined function.
gaussdb=# CREATE OR REPLACE FUNCTION int_add(int,int)
    RETURNS int AS $BODY$
declare
begin
    return $1 + $2;
end;
$BODY$ language plpgsql;

-- Create an aggregate function.
gaussdb=# CREATE AGGREGATE myavg(int)
(
    sfunc = int_add,
    stype = int,
    initcond = '0'
);
```



```
-- Delete the aggregate function myavg of the int type.
gaussdb=# DROP AGGREGATE myavg(int);

-- Delete the user-defined function.
gaussdb=# DROP FUNCTION int_add(int,int);
```

Helpful Links

ALTER AGGREGATE and **CREATE AGGREGATE**

Compatibility

The SQL standard does not provide the **DROP AGGREGATE** statement.

7.12.9.6 DROP AUDIT POLICY

Description

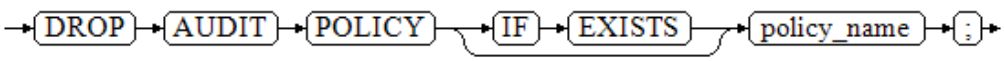
Deletes an audit policy.

Precautions

Only users with the POLADMIN or SYSADMIN permission, or the initial user can perform this operation.

Syntax

```
DROP AUDIT POLICY [IF EXISTS] policy_name;
```



Parameters

- **policy_name**

  Specifies the audit policy name, which must be unique.

  Value range: a string. It must comply with the **naming convention**.

- **IF EXISTS**

  Checks whether the audit policy exists. If it exists, the deletion is successful. Otherwise, a NOTICE message is sent.

Examples

```
-- Create the adt1 policy.
gaussdb=# CREATE AUDIT POLICY adt1 PRIVILEGES CREATE;
CREATE AUDIT POLICY

-- Delete the audit policy adt1.
gaussdb=# DROP AUDIT POLICY adt1;
DROP AUDIT POLICY

-- When you delete the audit policy adt0 that does not exist, the system displays a message indicating that the deletion fails because the audit policy does not exist.
gaussdb=# DROP AUDIT POLICY adt0;
ERROR:  adt0 policy does not exist, drop failed
```


Helpful Links

ALTER AUDIT POLICY and **CREATE AUDIT POLICY**

7.12.9.7 DROP CAST

Description

Deletes a type conversion.

Precautions

To delete a type conversion, you must have permissions on the source or destination data type. This is the same permission as creating a type conversion.

Syntax

```
DROP CAST [ IF EXISTS ] (source_type AS target_type) [ CASCADE | RESTRICT ];
```

Parameters

- **IF EXISTS**

  Do not throw an error if the specified conversion does not exist. A notice is issued in this case.
- **source_type**

  Source data type in the type conversion.
- **target_type**

  Type of the target data in the type conversion.
- **CASCADE** | **RESTRICT**

  These keys have no effect because there is no dependency on type conversion.

Examples

For details, see **8.12.8.12-Examples** in section "CREATE CAST."

Helpful Links

CREATE CAST

Compatibility

DROP CAST complies with the SQL standard.

7.12.9.8 DROP CLIENT MASTER KEY

Description

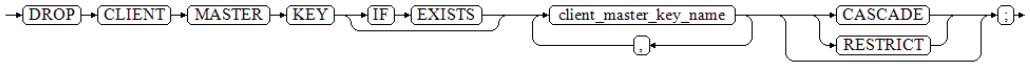
Deletes a CMK.

Precautions

- Only the CMK owner or a user who has been granted the DROP permission can run this command. By default, the system administrator has this permission.
- This command can only delete the metadata of key objects recorded in the system catalog of the database, but cannot delete the key entities managed by the client key tool or online key service.

Syntax

```
DROP CLIENT MASTER KEY [ IF EXISTS ] client_master_key_name [, ...] [ CASCADE | RESTRICT ];
```



Parameters

- **IF EXISTS**

  If a specified CMK does not exist, a notice rather than an error is issued.
- **client_master_key_name**

  Name of a CMK to be deleted.

  Value range: a string. It is the name of an existing CMK object.
- **CASCADE | RESTRICT**
  - Allows/Restricts to cascadingly delete objects that depend on the CMK.

Helpful Links

CREATE CLIENT MASTER KEY

7.12.9.9 DROP COLUMN ENCRYPTION KEY

Description

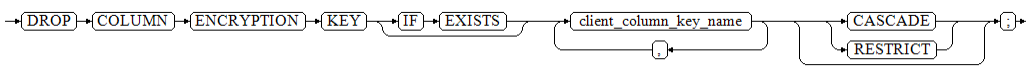
Deletes a column encryption key (CEK).

Precautions

Only the CEK owner or a user who has been granted the DROP permission can run this command. By default, the system administrator has this permission.

Syntax

```
DROP COLUMN ENCRYPTION KEY [ IF EXISTS ] client_column_key_name [, ...] [ CASCADE | RESTRICT ];
```



Parameters

- **IF EXISTS**

  If a specified CEK does not exist, a notice rather than an error is issued.

- **client_column_key_name**

  Name of a CEK to be deleted.

  Value range: a string. It is the name of an existing CEK.

- **CASCADE** | **RESTRICT**

  For fully-encrypted databases, this syntax is high-risk operation. Actually, encrypted columns that depend on CEKs cannot be deleted.

Helpful Links

ALTER COLUMN ENCRYPTION KEY and **CREATE COLUMN ENCRYPTION KEY**

7.12.9.10 DROP DATABASE

Description

Deletes a database.

Precautions

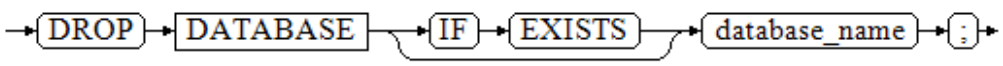
- Only the database owner or a user granted with the DROP permission can run the **DROP DATABASE** command. The system administrator has this permission by default.
- The preinstalled POSTGRES, TEMPLATE0, and TEMPLATE1 databases are protected and therefore cannot be deleted. To check databases in the current service, run the gsql command **\l**.
- If any users are connected to the database, the database cannot be deleted. You can view the DV_SESSIONS view to check the database connections.
- **DROP DATABASE** cannot be executed within a transaction block.
- If **DROP DATABASE** fails and is rolled back, run **DROP DATABASE IF EXISTS** again.

NOTICE

DROP DATABASE cannot be undone.

Syntax

```
DROP DATABASE [ IF EXISTS ] database_name;
```



Parameters

- **IF EXISTS**

  Reports a notice instead of an error if the specified database does not exist.

- **database_name**

  Specifies the name of the database to be deleted.

Value range: an existing database name

Examples

See **Examples** in section "CREATE DATABASE."

Helpful Links

ALTER DATABASE and **CREATE DATABASE**

Suggestions

- DROP DATABASE

  Do not delete databases during transactions.

7.12.9.11 DROP DATABASE LINK

Description

Deletes database link objects.

Syntax

```
DROP [ PUBLIC ] DATABASE LINK [ IF EXISTS ] dblink_name;
```

Parameters

- **dblink_name**

  Name of a connection object.

- **IF EXISTS**

  Reports a notice instead of an error if the specified **DATABASE LINK** does not exist.

- **PUBLIC**

  Specifies the connection type. If **PUBLIC** is not specified, the database link is private by default.

Examples

```
-- Create a user with the system administrator permission.
gaussdb=#  CREATE USER user1 WITH SYSADMIN PASSWORD '********';
gaussdb=#  SET ROLE user1 PASSWORD '********';

-- Create a private database link.
gaussdb=#  CREATE DATABASE LINK private_dblink CONNECT TO 'user1' IDENTIFIED BY '********' USING
(host '192.168.11.11',port '54399',dbname 'db01');

-- Delete the private database link.
gaussdb=#  DROP DATABASE LINK private_dblink;

-- Create a public database link.
gaussdb=#  CREATE PUBLIC DATABASE LINK public_dblink CONNECT TO 'user1' IDENTIFIED BY '********'
USING (host '192.168.11.11',port '54399',dbname 'db01');
```



```
-- Delete the public database link.
gaussdb=#  DROP PUBLIC DATABASE LINK public_dblink;

-- Delete the created user.
gaussdb=#  RESET ROLE;
gaussdb=#  DROP USER user1;
```

Helpful Links

ALTER DATABASE LINK and **CREATE DATABASE LINK**

7.12.9.12 DROP DIRECTORY

Description

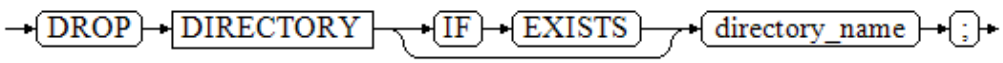
Deletes a directory.

Precautions

- When **enable_access_server_directory** is set to **off**, only the initial user is allowed to delete the directory object.
- When **enable_access_server_directory** is set to **on**, a user with the SYSADMIN permission, a directory object owner, a user granted with the DROP permission for a directory, or a user inherited from the built-in role gs_role_directory_drop can delete a directory object.

Syntax

```
DROP DIRECTORY [ IF EXISTS ] directory_name;
```



Parameters

- **directory_name**

  Specifies the name of the directory to be deleted.

  Value range: an existing directory name

- **IF EXISTS**

  If the directory object does not exist, a NOTICE message is displayed.

Examples

```
-- Create a directory object.
gaussdb=# CREATE OR REPLACE DIRECTORY  dir  as '/tmp/';

-- Delete the directory object.
gaussdb=# DROP DIRECTORY dir;
```

Helpful Links

CREATE DIRECTORY and **ALTER DIRECTORY**

7.12.9.13 DROP EVENT

Description

Deletes a scheduled task.

Precautions

Operations related to scheduled tasks are supported only when **sql_compatibility** is set to **'B'**.

Syntax

```
DROP EVENT [IF EXISTS] event_name;
```

Parameters

- **IF EXISTS**
  If the scheduled task does not exist, a NOTICE message is displayed.

Examples

```
-- Create and switch to the test database.
gaussdb=# CREATE DATABASE test_event WITH DBCOMPATIBILITY = 'b';
gaussdb=# \c test_event

-- Create a table.
test_event=# CREATE TABLE t_ev(num int);

-- Create a scheduled event that is executed once every five seconds. After the event is executed, the scheduled event is automatically deleted.
test_event=# CREATE EVENT IF NOT EXISTS event_e1 ON SCHEDULE AT sysdate + interval 5 second DO INSERT INTO t_ev VALUES(0);

-- Query the table five seconds later.
test_event=# SELECT * FROM t_ev;
 num
-----
   0
(1 row)

-- Create a scheduled event that is executed every minute.
test_event=# CREATE EVENT IF NOT EXISTS event_e2 ON SCHEDULE EVERY 1 minute DO INSERT INTO t_ev VALUES(1);

-- Query the table every one minute. A new record is displayed.
test_event=# SELECT * FROM t_ev;
 num
-----
   0
   1
   1
(3 rows)

-- Delete the scheduled event.
test_event=# DROP EVENT event_e2;

-- Drop the table.
test_event=# DROP TABLE t_ev;

-- Switch back to the initial database and delete the test database. Replace postgres with the actual database name.
test_event=#\c postgres
gaussdb=# DROP DATABASE test_event;
```


Helpful Links

ALTER EVENT, **CREATE EVENT**, and **SHOW EVENTS**

7.12.9.14 DROP EXTENSION

NOTICE

The extended function is for internal use only. You are advised not to use it.

Description

Deletes an extension.

Precautions

- The **DROP EXTENSION** command deletes an extension from the database. When you delete an extension, the components that make up the extension are also deleted.
- The **DROP EXTENSION** command can be used only by the owner of the extension.

Syntax

```
DROP EXTENSION [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ];
```

Parameters

- **IF EXISTS**

  If the **IF EXISTS** parameter is used and the extension does not exist, no error is reported. Instead, a notice is generated.
- **name**

  Specifies the name of an installed extension.
- **CASCADE**

  Automatically deletes objects that depend on the extension.
- **RESTRICT**

  If any object depends on an extension, the extension cannot be deleted (unless all its member objects and other extension objects are deleted at a time using the **DROP** command). This is a default processing.

Examples

Delete the PL/pgSQL extension from the current database. The PL/pgSQL extension is created by the database by default.

```
gaussdb=# DROP EXTENSION plpgsql;
```

In the current database, if an object that uses PL/pgSQL exists, this command will fail. For example, a column in any table is of the PL/pgSQL type. Adding the **CASCADE** option can forcibly delete the extension and objects that depend on it.

Helpful Links

ALTER EXTENSION and **CREATE EXTENSION**

7.12.9.15 DROP FOREIGN DATA WRAPPER

Description

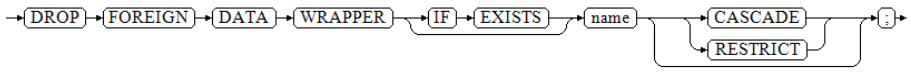
Deletes a specified foreign data wrapper.

Precautions

The DROP statement can be successfully executed only when **support_extended_features** is set to **on**.

Syntax

```
DROP FOREIGN DATA WRAPPER [ IF EXISTS ] name [ CASCADE | RESTRICT ];
```



Parameters

- **IF EXISTS**

  Sends a notification instead of throwing an error if the foreign data wrapper does not exist.
- **name**

  Specifies the name of an existing foreign data wrapper.
- **CASCADE**

  Automatically drops objects (such as servers) that depend on the foreign data wrapper.
- **RESTRICT**

  If there are objects that depend on the foreign data wrapper, the foreign data wrapper cannot be deleted. This is the default behavior.

Example

```
-- Delete the foreign data wrapper dbi.
gaussdb=# DROP FOREIGN DATA WRAPPER dbi;
```

Helpful Links

CREATE FOREIGN DATA WRAPPER and **ALTER FOREIGN DATA WRAPPER**

7.12.9.16 DROP FUNCTION

Description

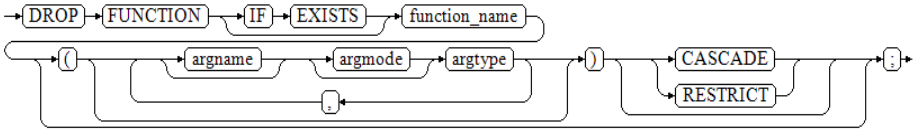
Deletes a function.

Precautions

- If a function involves operations on temporary tables, DROP FUNCTION cannot be used.
- Only the function owner or a user granted with the DROP permission can run the **DROP FUNCTION** command. The system administrator has this permission by default.

Syntax

```
DROP FUNCTION [ IF EXISTS ] function_name
[ ( [ {[ argname ] [ argmode ] argtype} [, ...] ] ) [ CASCADE | RESTRICT ] ];
```



Parameters

- **IF EXISTS**

  Reports a notice instead of an error if the specified function does not exist.
- **function_name**

  Specifies the name of the function to be deleted.

  Value range: an existing function name
- **argmode**

  Specifies the parameter mode of the function.
- **argname**

  Specifies the parameter name of the function.
- **argtype**

  Specifies the parameter type of the function.
- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes the objects that depend on the function.
  - **RESTRICT**: refuses to delete the function if any objects depend on it. This is the default action.

Examples

- Delete the list of parameters that can be omitted when a function.

```
-- Create a function.
gaussdb=# CREATE FUNCTION func_test(varchar) RETURN VARCHAR AS
BEGIN
    RETURN $1||_'test';
END;
/

-- Delete the function.
gaussdb=# DROP FUNCTION func_test;
```

- Delete a function with the same name.

  If a function with the same name exists, add a parameter list when deleting the function. Otherwise, an error is reported.

```
-- Create a function.
gaussdb=# CREATE FUNCTION func_add(int) RETURNS int AS $$
```



```
BEGIN
    RETURN $1+10;
END;
$$ LANGUAGE PLPGSQL;

-- Reload the func_add function.
gaussdb=# CREATE FUNCTION func_add(int,int) RETURNS int AS $$
BEGIN
    RETURN $1+$2;
END;
$$ LANGUAGE PLPGSQL;

-- Delete the function.
gaussdb=# DROP FUNCTION func_add(int);
gaussdb=# DROP FUNCTION func_add(int,int);
```

Helpful Links

ALTER FUNCTION and **CREATE FUNCTION**

7.12.9.17 DROP GLOBAL CONFIGURATION

Function

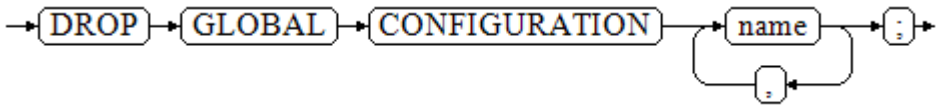
Deletes parameter values from the gs_global_config system catalog.

Precautions

- Only the initial database user can run this command.
- The parameter name cannot be **weak_password** or **undostoragetype**.

Syntax

```
DROP GLOBAL CONFIGURATION name [, ...];
```



Parameter Description

- **name**

  The parameter must exist in the gs_global_config system catalog. If you delete a parameter that does not exist, an error will be reported.

Examples

For details, see **Examples** in section "ALTER GLOBAL CONFIGURATION."

Helpful Links

ALTER GLOBAL CONFIGURATION

7.12.9.18 DROP GROUP

Description

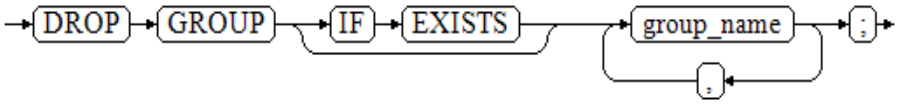
Deletes a user group. DROP GROUP is an alias for DROP ROLE.

Precautions

DROP GROUP is an API of the GaussDB management tool. You are advised not to use this API, because doing so affects GaussDB.

Syntax

```
DROP GROUP [ IF EXISTS ] group_name [, ...];
```



Parameters

- **IF EXISTS**

  If a role does not exist, no error is reported. Instead, a notification is sent, indicating that the role does not exist.

- **group_name**

  Specifies the name of the role to be deleted.

  Value range: an existing role name.

Examples

See **8.12.8.25 - Example** in section "CREATE GROUP."

Helpful Links

ALTER GROUP, **CREATE GROUP**, and **DROP ROLE**

7.12.9.19 DROP INDEX

Description

Deletes an index.

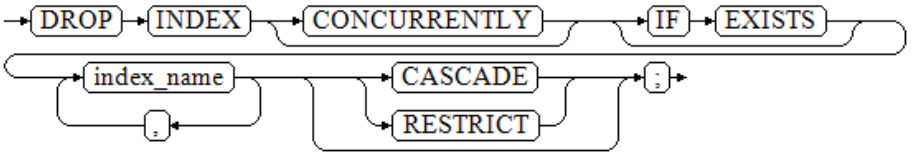
Precautions

- Only the index owner, owner of the schema where the index resides, a user who has the INDEX permission on the table where the index resides, or a user who has the DROP ANY INDEX permission can run the **DROP INDEX** command. When the separation of duties is disabled, the system administrator has this permission by default.
- For a global temporary table, if a session has initialized a global temporary table object (including creating a global temporary table and inserting data

into the global temporary table for the first time), other sessions cannot delete indexes from the table.

Syntax

```
DROP INDEX [ CONCURRENTLY ] [ IF EXISTS ]
    index_name [, ...] [ CASCADE | RESTRICT ];
```



Parameters

- **CONCURRENTLY**

  Deletes an index without locking it. When an index is deleted, other statements cannot access the table on which the index depends. With this option, the statement does not lock the table during index deletion.

  This parameter can specify only one index name and does not support **CASCADE**.

  The **DROP INDEX** statement can be run within a transaction, but **DROP INDEX CONCURRENTLY** cannot.

- **IF EXISTS**

  Reports a notice instead of an error if the specified index does not exist.

- **index_name**

  Specifies the index name to be deleted.

  Value range: an existing index

- **CASCADE | RESTRICT**
  - **CASCADE**: cascadingly deletes the objects that depend on the index.
  - **RESTRICT**: refuses to delete the index if any objects depend on it. This is the default action.

Examples

```
-- Create a table.
gaussdb=# CREATE TABLE test1_index (id INT, name VARCHAR(20));

-- Create an index.
gaussdb=# CREATE INDEX idx_test1 (id);

-- Delete the index.
gaussdb=# DROP INDEX IF EXISTS idx_test1 CASCADE;

-- Delete the table.
gaussdb=#  DROP TABLE test1_index;
```

Helpful Links

ALTER INDEX and **CREATE INDEX**

7.12.9.20 DROP LANGUAGE

This version does not support this syntax.

7.12.9.21 DROP MASKING POLICY

Description

Deletes a masking policy.

Precautions

Only users with the POLADMIN or SYSADMIN permission, or the initial user can perform this operation.

Syntax

```
DROP MASKING POLICY [ IF EXISTS ] policy_name;
```

Parameters

- **policy_name**

  Specifies the audit policy name, which must be unique.

  Value range: A string indicating an existing policy name.

Examples

```
-- Create table tb_for_masking.
gaussdb=# CREATE TABLE tb_for_masking(idx int, col1 text, col2 text, col3 text, col4 text, col5 text, col6
text, col7 text,col8 text);

-- Insert data into the tb_for_masking table.
gaussdb=# INSERT INTO tb_for_masking VALUES(1, '9876543210', 'usr321usr', 'abc@huawei.com',
'abc@huawei.com', '1234-4567-7890-0123', 'abcdef 123456 ui 323 jsfd321 j3k2l3', '4880-9898-4545-2525',
'this is a llt case');

-- Create a resource label for sensitive column col1.
gaussdb=# CREATE RESOURCE LABEL mask_lb1 ADD COLUMN(tb_for_masking.col1);

-- Create a resource label for sensitive column col2.
gaussdb=# CREATE RESOURCE LABEL mask_lb2 ADD COLUMN(tb_for_masking.col2);

-- Create a resource label for sensitive column col3.
gaussdb=# CREATE RESOURCE LABEL mask_lb3 ADD COLUMN(tb_for_masking.col3);

-- Create a data masking policy named maskpol1.
gaussdb=# CREATE MASKING POLICY maskpol1 maskall ON LABEL(mask_lb1);

-- Create a data masking policy named maskpol2.
gaussdb=# CREATE MASKING POLICY maskpol2 randommasking ON LABEL(mask_lb2);

-- Create a data masking policy named maskpol3.
gaussdb=# CREATE MASKING POLICY maskpol3 basicemailmasking ON LABEL(mask_lb3);

-- Delete a masking policy.
gaussdb=# DROP MASKING POLICY IF EXISTS maskpol1;

-- Delete a group of masking policies.
gaussdb=# DROP MASKING POLICY IF EXISTS maskpol2, maskpol3;

-- Delete a resource label.
```



```
gaussdb=# DROP RESOURCE LABEL mask_lb1;
gaussdb=# DROP RESOURCE LABEL mask_lb2;
gaussdb=# DROP RESOURCE LABEL mask_lb3;

-- Delete the table.
gaussdb=# DROP TABLE tb_for_masking;
```

Helpful Links

ALTER MASKING POLICY and **CREATE MASKING POLICY**

7.12.9.22 DROP MATERIALIZED VIEW

Description

Deletes an existing materialized view from the database.

Precautions

The owner of a materialized view, owner of the schema of the materialized view, users granted with the DROP permission on the materialized view, or users granted with the DROP ANY TABLE permission can run the **DROP MATERIALIZED VIEW** command. By default, the system administrator has the permission to run the command.

Syntax

```
DROP MATERIALIZED VIEW [ IF EXISTS ] mv_name [, ...] [ CASCADE | RESTRICT ];
```

Parameters

- **IF EXISTS**

  Reports a notice instead of an error if the specified materialized view does not exist.

- **mv_name**

  Name of the materialized view to be deleted.

- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes the objects that depend on the materialized view.
  - **RESTRICT**: refuses to delete the materialized view if any objects depend on it. This is the default action.

Examples

```
-- Create a table.
gaussdb=# CREATE TABLE my_table (c1 int, c2 int)
WITH(STORAGE_TYPE=ASTORE);

-- Create a materialized view named my_mv.
gaussdb=# CREATE MATERIALIZED VIEW my_mv AS SELECT * FROM my_table;

-- Delete the materialized view named my_mv.
```



```
gaussdb=# DROP MATERIALIZED VIEW my_mv;

-- Delete the table.
gaussdb=# DROP TABLE my_table;
```

Helpful Links

ALTER MATERIALIZED VIEW, **CREATE INCREMENTAL MATERIALIZED VIEW**, **CREATE MATERIALIZED VIEW**, **CREATE TABLE**, **REFRESH INCREMENTAL MATERIALIZED VIEW**, and **REFRESH MATERIALIZED VIEW**

7.12.9.23 DROP MODEL

Function

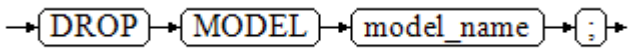
Deletes a model object that has been trained and saved.

Precautions

The deleted model can be viewed in the gs_model_warehouse system catalog.

Syntax

```
DROP MODEL model_name;
```



Parameter Description

- **model_name**

  Specifies a model name.

  Value range: a string. It must comply with the **naming convention**.

Examples

For details, see **8.12.8.31-Examples** in section "CREATE MODEL."

Helpful Links

CREATE MODEL and **PREDICT BY**

7.12.9.24 DROP OPERATOR

Deletes the definition of an operator.

Syntax

```
DROP OPERATOR [ IF EXISTS ]  name ( { left_type | NONE } , { right_type | NONE } );
```

Description

- **name**

  Name of an existing operator.

- **left_type**

  Data type of the left operand for the operator; if there is no left operand, write NONE.

- **right_type**

  Data type of the right operand for the operator; if there is no right operand, write NONE.

Examples

For details, see **8.12.8.32-Examples** in section "CREATE OPERATOR."

Helpful Links

ALTER OPERATOR, **CREATE OPERATOR**, and **CREATE OPERATOR CLASS**

7.12.9.25 DROP OWNED

Function

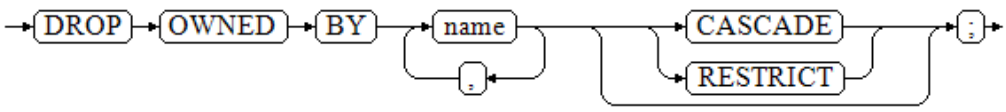
Deletes permissions on database objects owned by a database role.

Precautions

- This interface will revoke the role's permissions on all objects in the current database and shared objects (databases and tablespaces).
- DROP OWNED is often used to prepare for removing one or more roles. Because DROP OWNED affects only the objects in the current database, you need to run this statement in each database that contains the objects owned by the role to be removed.
- Using the **CASCADE** option may cause this statement to recursively remove objects owned by other users.
- The databases and tablespaces owned by the role will not be removed.
- Private database links owned by a role can be deleted only after the **CASCADE** option is added.

Syntax

```
DROP OWNED BY name [, ...] [ CASCADE | RESTRICT ];
```



Parameter Description

- **name**

  Specifies the role name.

- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes the objects that depend on the objects to be deleted.

  - **RESTRICT**: refuses to delete the object if any objects depend on it. This is the default action.

Examples

```
-- Create user jim.
gaussdb=# CREATE USER jim PASSWORD '********';

-- Revoke user jim's permissions on all objects in the current database and on shared objects (databases and tablespaces).
gaussdb=# DROP OWNED BY jim;

-- Delete user jim.
gaussdb=# DROP USER jim;
```

Helpful Links

REASSIGN OWNED and **DROP ROLE**

7.12.9.26 DROP PACKAGE

Description

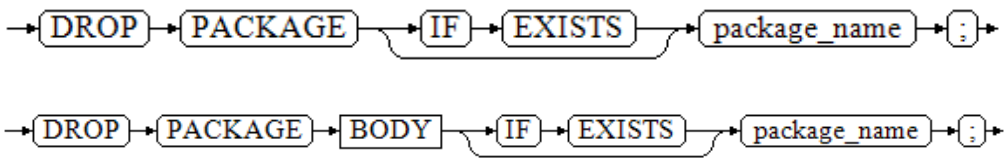
Deletes the existing package or package body.

Precautions

After the package body is deleted, the stored procedures and functions in the package become invalid at the same time.

Syntax

```
DROP PACKAGE [ IF EXISTS ] package_name;
DROP PACKAGE BODY [ IF EXISTS ] package_name;
```



Parameters

- **IF EXISTS**

  Reports a notice instead of an error if the specified stored procedure does not exist.

- **package_name**

  Specifies the name of the package to be deleted.

  Value range: an existing package name.

Examples

```
-- Create a package.
gaussdb=# CREATE OR REPLACE PACKAGE PCK1
IS
a int;
```



```
END pck1;
/
CREATE PACKAGE

-- Delete the package.
gaussdb=# DROP PACKAGE PCK1;
DROP PACKAGE
```

Helpful Links

ALTER PACKAGE and **CREATE PACKAGE**

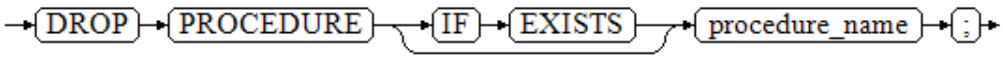
7.12.9.27 DROP PROCEDURE

Function

Deletes a stored procedure.

Syntax

```
DROP PROCEDURE [ IF EXISTS ] procedure_name;
```



Parameter Description

- **IF EXISTS**

  Reports a notice instead of an error if the specified stored procedure does not exist.

- **procedure_name**

  Specifies the name of the stored procedure to be deleted.

  Value range: an existing stored procedure name

Examples

For details, see **8.12.8.35-Examples** in section "CREATE PROCEDURE."

Helpful Links

ALTER PROCEDURE and **CREATE PROCEDURE**

7.12.9.28 DROP RESOURCE LABEL

Description

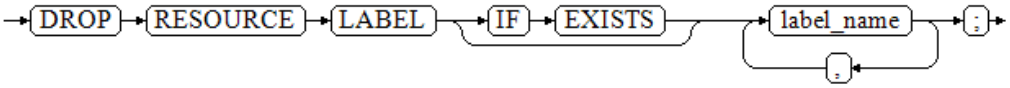
DROP RESOURCE LABEL is used to delete resource labels.

Precautions

Only users with the POLADMIN or SYSADMIN permission, or the initial user can perform this operation.

Syntax

```
DROP RESOURCE LABEL [IF EXISTS] label_name[, ...];
```



Parameters

- **label_name**

  Specifies the resource label name.

  Value range: a string. It must comply with the **naming convention**.

Examples

```
-- Create table tb_for_label.
gaussdb=# CREATE TABLE tb_for_label(col1 text, col2 text, col3 text);

-- Create a resource label based on a table.
gaussdb=# CREATE RESOURCE LABEL IF NOT EXISTS table_label add TABLE(public.tb_for_label);

-- Create an existing table resource label again and compare the differences between adding IF NOT EXISTS and not adding IF NOT EXISTS.
gaussdb=# CREATE RESOURCE LABEL IF NOT EXISTS table_label add TABLE(public.tb_for_label);
NOTICE:  table_label label already defined, skipping
CREATE RESOURCE LABEL
gaussdb=# CREATE RESOURCE LABEL table_label add TABLE(public.tb_for_label);
ERROR:  table_label label already defined

-- Create a resource label based on columns.
gaussdb=# CREATE RESOURCE LABEL IF NOT EXISTS column_label add COLUMN(public.tb_for_label.col1);

-- Create schema schema_for_label.
gaussdb=# CREATE SCHEMA schema_for_label;

-- Create a resource label based on a schema.
gaussdb=# CREATE RESOURCE LABEL IF NOT EXISTS schema_label add SCHEMA(schema_for_label);

-- Create view view_for_label.
gaussdb=# CREATE VIEW view_for_label AS SELECT 1;

-- Create a resource label based on a view.
gaussdb=# CREATE RESOURCE LABEL IF NOT EXISTS view_label add VIEW(view_for_label);

-- Create function func_for_label.
gaussdb=# CREATE FUNCTION func_for_label RETURNS TEXT AS $$ SELECT col1 FROM tb_for_label; $$ LANGUAGE SQL;

-- Create a resource label based on a function.
gaussdb=# CREATE RESOURCE LABEL IF NOT EXISTS func_label add FUNCTION(func_for_label);

-- Delete the table resource label table_label.
gaussdb=# DROP RESOURCE LABEL IF EXISTS table_label;

-- Delete the column resource label column_label.
gaussdb=# DROP RESOURCE LABEL IF EXISTS column_label;

-- Delete the function resource label func_for_label.
gaussdb=# DROP FUNCTION func_for_label;

-- Delete the view resource label view_for_label.
gaussdb=# DROP VIEW view_for_label;

-- Delete the schema resource label schema_for_label.
gaussdb=# DROP SCHEMA schema_for_label;
```



```
-- Delete the tb_for_label table.
gaussdb=# DROP TABLE tb_for_label;
```

Helpful Links

ALTER RESOURCE LABEL and **CREATE RESOURCE LABEL**

7.12.9.29 DROP RESOURCE POOL

Description

Deletes a resource pool.

NOTE

The resource pool cannot be deleted if it is associated with a role.

Precautions

Only users with the SYSADMIN permission, or the initial user can perform this operation.

Syntax

```
DROP RESOURCE POOL [ IF EXISTS ] pool_name;
```

Parameters

- **IF EXISTS**

  Reports a notice instead of an error if a specified resource pool does not exist.

- **pool_name**

  Specifies the name of a created resource pool.

  Value range: a string. It must comply with the **naming convention**.

Examples

See **Examples** in section "CREATE RESOURCE POOL."

Helpful Links

ALTER RESOURCE POOL and **CREATE RESOURCE POOL**

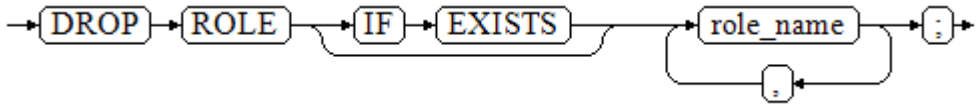
7.12.9.30 DROP ROLE

Function

Deletes a role.

Syntax

```
DROP ROLE [ IF EXISTS ] role_name [, ...];
```



Parameter Description

- **IF EXISTS**

  Reports a notice instead of an error if the specified role does not exist.
- **role_name**

  Specifies the name of the role to be deleted.

  Value range: an existing role name

Examples

See **Examples** in section "CREATE ROLE."

Helpful Links

CREATE ROLE, **ALTER ROLE**, and **SET ROLE**

7.12.9.31 DROP ROW LEVEL SECURITY POLICY

Function

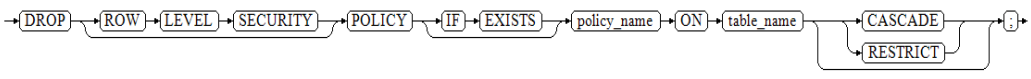
Deletes a row-level security policy from a table.

Precautions

Only the table owner or administrators can delete a row-level security policy from the table.

Syntax

```
DROP [ ROW LEVEL SECURITY ] POLICY [ IF EXISTS ] policy_name ON table_name [ CASCADE | RESTRICT ];
```



Parameter Description

- **IF EXISTS**

  Reports a notice instead of an error if the specified row-level security policy does not exist.
- **policy_name**

  Specifies the name of the row-level security policy to be deleted.
- **table_name**

  Specifies the name of the table containing the row-level security policy.

- **CASCADE | RESTRICT**

  Currently, no objects depend on row-level security policies. Therefore, CASCADE is equivalent to RESTRICT, and they are reserved to ensure backward compatibility.

Examples

```
-- Create the data table all_data.
gaussdb=# CREATE TABLE all_data(id int, role varchar(100), data varchar(100));

-- Create a row-level security policy.
gaussdb=# CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role = CURRENT_USER);

-- Delete the row-level security policy.
gaussdb=# DROP ROW LEVEL SECURITY POLICY all_data_rls ON all_data;

-- Delete the all_data table.
gaussdb=# DROP TABLE all_data;
```

Helpful Links

ALTER ROW LEVEL SECURITY POLICY and **CREATE ROW LEVEL SECURITY POLICY**

7.12.9.32 DROP RULE

Function

Deletes a rewriting rule.

Syntax

```
DROP RULE [ IF EXISTS ] name ON table_name [ CASCADE | RESTRICT ];
```

Parameter Description

- **IF EXISTS**

  If the rule does not exist, a notice is thrown.

- **name**

  Name of an existing rule to be deleted.

- **table_name**

  Name of the table to which the rule applies.

- **CASCADE**

  Automatically cascade deletes objects that depend on this rule.

- **RESTRICT**

  Refuses to delete the rule if any objects depend on it. The default value is used.

Examples

```
-- Create the def_test table and the def_view_test view for creating rules.
gaussdb=# CREATE TABLE def_test (
c1 int4 DEFAULT 5,
c2 text DEFAULT 'initial_default'
);
```



```
gaussdb=# CREATE VIEW def_view_test AS SELECT * FROM def_test;
-- Create the rule def_view_test_ins.
gaussdb=# CREATE RULE def_view_test_ins AS
gaussdb=# ON INSERT TO def_view_test
gaussdb=#DO INSTEAD INSERT INTO def_test SELECT new.*;
-- Delete the rule def_view_test_ins.
gaussdb=# DROP RULE def_view_test_ins ON def_view_test;
-- Delete the def_test table and the def_view_test view.
gaussdb=# DROP VIEW def_view_test;
gaussdb=# DROP TABLE def_test;
```

Helpful Links

CREATE RULE

7.12.9.33 DROP SCHEMA

Description

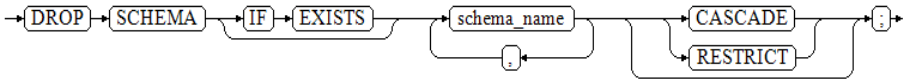
Deletes a schema from the current database.

Precautions

- Only the owner of a schema or a user granted the DROP permission for a schema has the permission to run the **DROP SCHEMA** command. If separation of duties is disabled, the system administrator has this permission by default.
- Users except the initial user and O&M administrator cannot drop the schema of the O&M administrator.
- You are not allowed to delete DBE_PLDEVELOPER when **allow_system_table_mods** is disabled.

Syntax

```
DROP SCHEMA [ IF EXISTS ] schema_name [, ...] [ CASCADE | RESTRICT ];
```



Parameters

- **IF EXISTS**

  Reports a notice instead of an error if the specified schema does not exist.
- **schema_name**

  Specifies the name of the schema to be deleted.

  Value range: an existing schema name
- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes all the objects contained in the schema.
  - **RESTRICT**: refuses to delete the schema if the schema contains objects. This is the default action.

NOTICE

Schemas beginning with **pg_temp** or **pg_toast_temp** are for internal use. Do not delete them. Otherwise, unexpected consequences may be incurred.

NOTE

The schema currently being used cannot be deleted. To delete it, switch to another schema first.

Examples

See **Examples** in section "CREATE SCHEMA."

Helpful Links

ALTER SCHEMA and **CREATE SCHEMA**

7.12.9.34 DROP SECURITY LABEL

Description

DROP SECURITY LABEL is used to delete security labels from the current database.

Precautions

An initial user, a user with the SYSADMIN permission, or a user who inherits the gs_role_seclabel permission of the built-in role can delete security labels.

Syntax

```
DROP SECURITY LABEL label_name;
```

Parameters

- **label_name**

  Security label name. The security label to be deleted must exist in the database.

Examples

```
-- Create a security label sec_label.
gaussdb=# CREATE SECURITY LABEL sec_label 'L1:G4';

-- Delete the security label sec_label2 that does not exist.
gaussdb=# DROP SECURITY LABEL sec_label2;
ERROR:  security label "sec_label2" does not exist

-- Delete the existing security label sec_label.
gaussdb=# DROP SECURITY LABEL sec_label;
```

Helpful Links

CREATE SECURITY LABEL and **SECURITY LABEL ON**

7.12.9.35 DROP SEQUENCE

Description

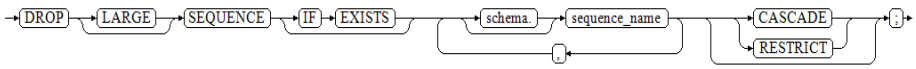
Deletes a sequence from the current database.

Precautions

- Only the owner of a sequence, the owner of the schema to which the sequence belongs, or a user granted the DROP permission on a sequence or a user granted the DROP ANY SEQUENCE permission can delete a sequence. When separation of duties is disabled, a system administrator has this permission by default.
- If the LARGE identifier is used when a sequence is created, the LARGE identifier must be used when the sequence is dropped.

Syntax

```
DROP [ LARGE ] SEQUENCE [ IF EXISTS ] {[schema.]sequence_name} [ , ... ] [ CASCADE | RESTRICT ];
```



Parameters

- **IF EXISTS**

  Reports a notice instead of an error if the specified sequence does not exist.
- **sequence_name**

  Specifies the name of the sequence to be deleted.
- **CASCADE**

  Cascadingly deletes the objects that depend on the sequence.
- **RESTRICT**

  Refuses to delete the sequence if any objects depend on it. This is the default action.

Examples

```
-- Create an ascending sequence named serial, starting from 101.
gaussdb=# CREATE SEQUENCE serial START 101;

-- Delete a sequence.
gaussdb=# DROP SEQUENCE serial;
```

Helpful Links

ALTER SEQUENCE and **DROP SEQUENCE**

7.12.9.36 DROP SERVER

Description

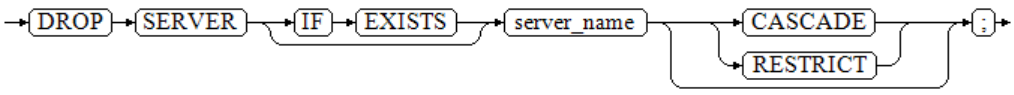
Deletes an existing data server.

Precautions

Only the server owner or a user granted with the DROP permission can run the **DROP SERVER** command. A system administrator has this permission by default.

Syntax

```
DROP SERVER [ IF EXISTS ] server_name [ CASCADE | RESTRICT ] ;
```



Parameters

- **IF EXISTS**

  Reports a notice instead of an error if the specified data server does not exist.
- **server_name**

  Specifies the name of the data server to be dropped.
- **CASCADE | RESTRICT**
  - **CASCADE**: cascadingly deletes the objects that depend on the data server.
  - **RESTRICT**: refuses to delete the server if any objects depend on it. This is the default action.

Example

```
-- Create a server.
gaussdb=# CREATE SERVER my_server FOREIGN DATA WRAPPER log_fdw;
CREATE SERVER

-- Delete the server.
gaussdb=# DROP SERVER my_server;
DROP SERVER
```

Helpful Links

ALTER SERVER and **CREATE SERVER**

7.12.9.37 DROP SYNONYM

Description

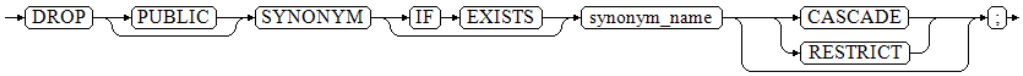
Deletes a synonym.

Precautions

Only the synonym owner or a user granted with the DROP ANY SEQUENCE permission can run the **DROP SYNONYM** command. The system administrator has this permission by default.

Syntax

```
DROP [PUBLIC] SYNONYM [ IF EXISTS ] synonym_name [ CASCADE | RESTRICT ];
```

Parameters

- **PUBLIC**

  (Optional) Deletes PUBLIC synonyms.

- **IF EXISTS**

  Reports a notice instead of an error if the specified synonym does not exist.

- **synonym_name**

  Specifies the name (optionally schema-qualified) of the synonym to be deleted.

- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes the objects (such as views) that depend on the synonym.
  - **RESTRICT**: refuses to delete the synonym if any objects depend on it. This is the default action.

Examples

See **Examples** in section "CREATE SYNONYM."

Helpful Links

ALTER SYNONYM and **CREATE SYNONYM**

7.12.9.38 DROP TABLE

Description

Deletes a table.

Precautions

- After DROP TABLE deletes the table, the indexes depending on the table are deleted, and the functions and stored procedures that need to use this table cannot be executed. Deleting a partitioned table also deletes all partitions in the table.
- The owner of a table, the owner of the schema of the table, users granted with the DROP permission on the table, or users granted with the DROP ANY TABLE permission can delete the specified table. When separation of duties is disabled, a system administrator has the permission to delete the specified table by default.
- When DROP TABLE is executed, if the table to be deleted references another table as a foreign key table, triggers on the referenced table will be deleted in cascading mode. In this case, an eight-level lock needs to be added to the referenced table, which may cause service congestion.

Syntax

```
DROP TABLE [ IF EXISTS ]
    { [schema.]table_name } [, ...] [ CASCADE | RESTRICT ] [ PURGE ];
```

Parameters

- **IF EXISTS**

  Reports a notice instead of an error if the specified table does not exist.

- **schema**

  Specifies the schema name.

- **table_name**

  Specifies the name of the table to be deleted.

- **CASCADE | RESTRICT**
  - **CASCADE**: allows to cascadingly delete the objects (such as views) that depend on the table.
  - **RESTRICT**: refuses to delete the table if any objects depend on it. This is the default action.

- **PURGE**

  Specifies that even if the recycle bin function is enabled, the table is physically dropped instead of being moved to the recycle bin when you use DROP TABLE to delete tables.

Examples

```
-- Create the test table.
gaussdb=# CREATE TABLE test(c1 int, c2 int);
-- Drop the test table.
gaussdb=# DROP TABLE IF EXISTS test;

-- Create the test1 table.
gaussdb=# CREATE TABLE test1(c1 int, c2 int);
-- Create the v_test1 view.
gaussdb=# CREATE VIEW v_test1 AS SELECT * FROM test1 WHERE c1 < 20;
-- An error is reported when the table is deleted.
gaussdb=# DROP TABLE test1;
ERROR:  cannot drop table test1 because other objects depend on it
DETAIL:  view v_test1 depends on table test1
HINT:  Use DROP ... CASCADE to drop the dependent objects too.

-- Use the CASCADE parameter to drop the test1 table, which will automatically delete the views.
gaussdb=# DROP TABLE test1 CASCADE;
NOTICE:  drop cascades to view v_test1
DROP TABLE
```

Helpful Links

ALTER TABLE and **CREATE TABLE**

7.12.9.39 DROP TABLESPACE

Description

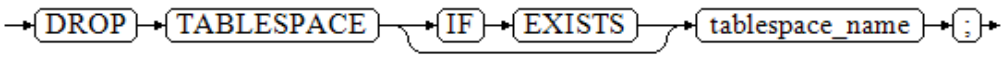
Deletes a tablespace.

Precautions

- Only the tablespace owner or a user granted with the DROP permission can run the **DROP TABLESPACE** command. The system administrator has this permission by default.
- The tablespace to be deleted should not contain any database objects. Otherwise, an error will be reported.
- DROP TABLESPACE cannot be rolled back and therefore cannot be run in transaction blocks.
- During execution of DROP TABLESPACE, database queries by other sessions using **\db** may fail and need to be reattempted.
- If DROP TABLESPACE fails to be executed, execute DROP TABLESPACE IF EXISTS.

Syntax

```
DROP TABLESPACE [ IF EXISTS ] tablespace_name;
```



Parameters

- **IF EXISTS**

  Reports a notice instead of an error if the specified tablespace does not exist.
- **tablespace_name**

  Specifies the name of the tablespace to be deleted.

  Value range: an existing tablespace name

Examples

See **Examples** in section "CREATE TABLESPACE."

Helpful Links

ALTER TABLESPACE and **CREATE TABLESPACE**

Suggestions

- DROP TABLESPACE

  Do not delete tablespaces during transactions.

7.12.9.40 DROP TRIGGER

Description

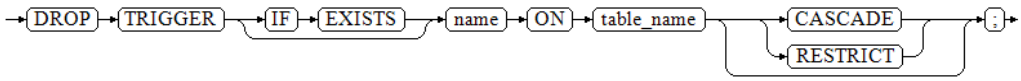
DROP TRIGGER is used to delete a trigger.

Precautions

Only the trigger owner or a user granted with the DROP ANY TRIGGER permission can run the **DROP TRIGGER** command. The system administrator has this permission by default.

Syntax

```
DROP TRIGGER [ IF EXISTS ] name ON table_name [ CASCADE | RESTRICT ];
```



Parameters

- **IF EXISTS**

  Reports a notice instead of an error if the specified trigger does not exist.
- **trigger_name**

  Specifies the name of the trigger to be deleted.

  Value range: an existing trigger name
- **table_name**

  Specifies the name of the table containing the trigger.

  Value range: name of the table containing the trigger
- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes the objects that depend on the trigger.
  - **RESTRICT**: refuses to delete the trigger if any objects depend on it. This is the default action.

Examples

```
-- Create a source table and a target table.
gaussdb=# CREATE TABLE test_trigger_src_tbl(id1 INT, id2 INT, id3 INT);
gaussdb=# CREATE TABLE test_trigger_des_tbl(id1 INT, id2 INT, id3 INT);
```

Use the INSERT trigger.

```
-- Create an INSERT trigger function.
gaussdb=# CREATE OR REPLACE FUNCTION tri_insert_func() RETURNS TRIGGER AS
        $$
        DECLARE
        BEGIN
                INSERT INTO test_trigger_des_tbl VALUES(NEW.id1, NEW.id2, NEW.id3);
                RETURN NEW;
        END
        $$ LANGUAGE plpgsql;

-- Create an INSERT trigger.
gaussdb=# CREATE TRIGGER insert_trigger
        BEFORE INSERT ON test_trigger_src_tbl
        FOR EACH ROW
        EXECUTE PROCEDURE tri_insert_func();

-- Execute the INSERT event and check the trigger results.
gaussdb=# INSERT INTO test_trigger_src_tbl VALUES(100,200,300);

gaussdb=# SELECT * FROM test_trigger_src_tbl;
 id1 | id2 | id3
```



```
-----+-----+-----
 100 | 200 | 300
(1 row)

gaussdb=# SELECT * FROM test_trigger_des_tbl; // Check whether the trigger operation takes effect.
 id1 | id2 | id3
-----+-----+-----
 100 | 200 | 300
(1 row)
```

Use anonymous blocks and the OR REPLACE syntax to create triggers.

```
-- Create an INSERT trigger using the anonymous block syntax.
gaussdb=# CREATE TRIGGER insert_trigger_with_anonyblock
        BEFORE INSERT ON test_trigger_src_tbl
        FOR EACH ROW
        DECLARE
        BEGIN
                INSERT INTO test_trigger_des_tbl VALUES(NEW.id1, NEW.id2, NEW.id3);
                RETURN NEW;
        END;
        /

-- Create an INSERT trigger using the OR REPLACE syntax.
gaussdb=# CREATE OR REPLACE TRIGGER insert_trigger_with_anonyblock
        BEFORE INSERT ON test_trigger_src_tbl
        FOR EACH ROW
        DECLARE
        BEGIN
                INSERT INTO test_trigger_des_tbl VALUES(NEW.id1, NEW.id2, NEW.id3);
                RETURN NEW;
        END;
        /

-- Delete the trigger.
gaussdb=# DROP TRIGGER insert_trigger_with_anonyblock ON test_trigger_src_tbl; -- Delete the implicitly
created function insert_trigger_with_anonyblock.
```

Use the UPDATE trigger.

```
-- Create an UPDATE trigger function.
gaussdb=# CREATE OR REPLACE FUNCTION tri_update_func() RETURNS TRIGGER AS
$$
DECLARE
BEGIN
UPDATE test_trigger_des_tbl SET id3 = NEW.id3 WHERE id1=OLD.id1;
RETURN OLD;
END
$$ LANGUAGE plpgsql;

-- Create an UPDATE trigger.
gaussdb=# CREATE TRIGGER update_trigger
        AFTER UPDATE ON test_trigger_src_tbl
        FOR EACH ROW
        EXECUTE PROCEDURE tri_update_func();

-- Execute the UPDATE event and check the trigger results.
gaussdb=# UPDATE test_trigger_src_tbl SET id3=400 WHERE id1=100;

gaussdb=# SELECT * FROM test_trigger_src_tbl;
 id1 | id2 | id3
-----+-----+-----
 100 | 200 | 400
(1 row)

gaussdb=# SELECT * FROM test_trigger_des_tbl; // Check whether the trigger operation takes effect.
 id1 | id2 | id3
-----+-----+-----
 100 | 200 | 400
(1 row)
```


Use the DELETE trigger.

```
-- Create a DELETE trigger function.
gaussdb=# CREATE OR REPLACE FUNCTION TRI_DELETE_FUNC() RETURNS TRIGGER AS
        $$
        DECLARE
        BEGIN
                DELETE FROM test_trigger_des_tbl WHERE id1=OLD.id1;
                RETURN OLD;
        END
        $$ LANGUAGE plpgsql;

-- Create a DELETE trigger.
gaussdb=# CREATE TRIGGER delete_trigger BEFORE DELETE ON test_trigger_src_tbl FOR EACH ROW
EXECUTE PROCEDURE tri_delete_func();

-- Execute the DELETE event and check the trigger results.
gaussdb=# DELETE FROM test_trigger_src_tbl WHERE id1=100;

gaussdb=# SELECT * FROM test_trigger_src_tbl;
 id1 | id2 | id3
-----+-----+-----
(0 rows)

gaussdb=# SELECT * FROM test_trigger_des_tbl; // Check whether the trigger operation takes effect.
 id1 | id2 | id3
-----+-----+-----
(0 rows)
```

Rename a trigger.

```
-- Rename a trigger.
gaussdb=# ALTER TRIGGER delete_trigger ON test_trigger_src_tbl RENAME TO delete_trigger_renamed;
```

Disable a trigger.

```
-- Disable insert_trigger.
gaussdb=# ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER insert_trigger;

gaussdb=# INSERT INTO test_trigger_src_tbl VALUES(100,200,300);

gaussdb=# SELECT * FROM test_trigger_src_tbl;
 id1 | id2 | id3
-----+-----+-----
 100 | 200 | 300
(1 row)

gaussdb=# SELECT * FROM test_trigger_des_tbl; // The trigger does not take effect.
 id1 | id2 | id3
-----+-----+-----
(0 rows)

-- Disable all triggers on the current table.
gaussdb=# ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER ALL;
```

Delete a trigger.

```
gaussdb=# DROP TRIGGER insert_trigger ON test_trigger_src_tbl;

gaussdb=# DROP TRIGGER update_trigger ON test_trigger_src_tbl;

gaussdb=# DROP TRIGGER delete_trigger_renamed ON test_trigger_src_tbl;
```

Delete a function.

```
gaussdb=# DROP FUNCTION tri_insert_func;

gaussdb=# DROP FUNCTION tri_update_func;

gaussdb=# DROP FUNCTION tri_delete_func;
```



```
-- Delete the source table and target table.
gaussdb=# DROP TABLE test_trigger_src_tbl;
gaussdb=# DROP TABLE test_trigger_des_tbl;
```

Helpful Links

CREATE TRIGGER, **ALTER TRIGGER**, and **ALTER TABLE**

7.12.9.41 DROP TYPE

Description

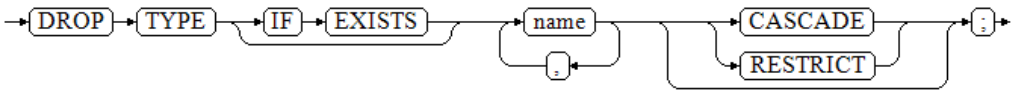
Deletes a user-defined data type.

Precautions

Only the owner of a type, a user granted the DROP permission on a type, or a user granted the DROP ANY TYPE permission on a sequence can run the **DROP TYPE** command. When separation of duties is disabled, a system administrator has this permission by default.

Syntax

```
DROP TYPE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ];
```



Parameters

- **IF EXISTS**

  Reports a notice instead of an error if the specified type does not exist.

- **name**

  Specifies the name (optionally schema-qualified) of the type to be deleted.

- **CASCADE**

  Automatically deletes the objects (such as columns, functions, and operators) that depend on the type.

- **RESTRICT**

  Refuses to delete the type if any objects depend on it. This is the default action.

Examples

See **Examples** in section "CREATE TYPE."

Helpful Links

CREATE TYPE and **ALTER TYPE**

7.12.9.42 DROP USER

Description

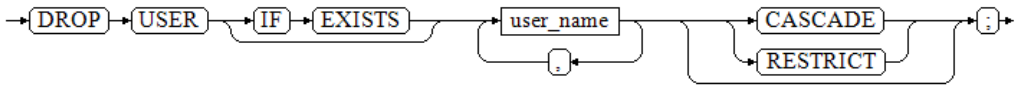
DROP USER deletes users in the GaussDB. This permission can be executed only when you have the permission to create users. After the command for deleting a user is executed successfully, the schema with the same name is deleted.

Precautions

- **CASCADE** is used to delete the objects (excluding databases) that depend on the user. It cannot delete locked objects unless the objects are unlocked or the threads locking the objects are terminated.
- If the dependent objects are other databases or reside in other databases, manually delete them before deleting the user from the current database. DROP USER cannot delete objects across databases.
- Before deleting a user, you need to delete all the objects owned by the user and revoke the user's permissions on other objects. Alternatively, you can specify **CASCADE** to delete the objects owned by the user and the granted permissions.

Syntax

```
DROP USER [ IF EXISTS ] user_name [, ...] [ CASCADE | RESTRICT ];
```



Parameters

- **IF EXISTS**

  When this parameter is used, if the specified user does not exist, a notice instead of an error is sent. Therefore, this parameter can be used to avoid errors.

- **user_name**

  Specifies the name of the user to be deleted.

  Value range: an existing username in the database.

- **CASCADE | RESTRICT**
  - **CASCADE**: automatically deletes the objects that depend on the user.
  - **RESTRICT**: refuses to delete the user if any objects depend on it. This is the default action.

    NOTE

    In GaussDB, the **enable_kill_query** configuration parameter exists in the **gaussdb.conf** file. This parameter affects **CASCADE**.

    - If **enable_kill_query** is **on** and **CASCADE** is used, the statement automatically kills the threads locking dependent objects and then deletes the specified user.
    - If **enable_kill_query** is **off** and **CASCADE** is used, the statement waits until the threads locking dependent objects end and then deletes the specified user.

Examples

```
-- Create user jim whose login password is ********.
gaussdb=# CREATE USER jim PASSWORD '********';

-- Create user kim whose login password is ********.
gaussdb=# CREATE USER kim IDENTIFIED BY '********';

-- Create user tom whose login password is ********.
gaussdb=# CREATE USER TOM PASSWORD '**********';

-- Create user TOM whose login password is ********.
gaussdb=# CREATE USER "TOM" PASSWORD '**********';

-- To create a user with the CREATEDB permission, add the CREATEDB keyword.
gaussdb=# CREATE USER dim CREATEDB PASSWORD '********';

-- Query the permissions of the dim user.
gaussdb=#  \du dim
          List of roles
 Role name | Attributes | Member of
-----------+------------+-----------
 dim       | Create DB  | {}
(You can see that the dim user has the CREATEDB permission.)

-- Change the login password of user jim.
gaussdb=# ALTER USER jim IDENTIFIED BY '**********' REPLACE '********';

-- Add the CREATEROLE permission to jim.
gaussdb=# ALTER USER jim CREATEROLE;

-- View the CREATEROLE permission added to user jim.
gaussdb=# \du jim
           List of roles
 Role name | Attributes  | Member of
-----------+-------------+-----------
 jim       | Create role | {}

-- Set enable_seqscan to on. (The setting will take effect in the next session.)
gaussdb=# ALTER USER jim SET enable_seqscan TO on;

-- Reset the enable_seqscan parameter for jim.
gaussdb=# ALTER USER jim RESET enable_seqscan;

-- Lock jim.
gaussdb=# ALTER USER jim ACCOUNT LOCK;

-- Unlock jim.
gaussdb=# ALTER USER jim ACCOUNT UNLOCK;

-- Change the user password.
gaussdb=# ALTER USER dim WITH PASSWORD '********';

-- Change the username.
gaussdb=# ALTER USER dim RENAME TO lisa;

-- Delete the user.
gaussdb=# DROP USER kim CASCADE;
gaussdb=# DROP USER jim CASCADE;
gaussdb=# DROP USER lisa CASCADE;
gaussdb=# DROP USER TOM CASCADE;
gaussdb=# DROP USER "TOM" CASCADE;
```

Helpful Links

ALTER USER and **CREATE USER**

7.12.9.43 DROP USER MAPPING

Description

DROP USER MAPPING is used to delete the user mapping from a user to a foreign server.

Syntax

```
DROP USER MAPPING [ IF EXISTS ] FOR { user_name | USER | CURRENT_USER | PUBLIC } SERVER
server_name;
```

Parameters

- **IF EXISTS**

  Reports a notice instead of an error if the user mapping does not exist.

  > ⚠ CAUTION
  >
  > Different from many statements, the **IF EXISTS** parameter is used only for the DROP USER MAPPING statement. When the CREATE USER MAPPING statement uses this parameter, a syntax error is reported.

- **user_name**

  Specifies username of the mapping.

  CURRENT_USER and USER match the name of the current user. PUBLIC is used to match all current and future usernames in the system.

- **server_name**

  Specifies name of the server to which the user is mapped.

Examples

```
-- Create a role.
gaussdb=# CREATE ROLE bob PASSWORD '********';

-- Create a foreign server.
gaussdb=# CREATE SERVER my_server FOREIGN DATA WRAPPER log_fdw;

-- Create a user mapping.
gaussdb=# CREATE USER MAPPING FOR bob SERVER my_server OPTIONS (USER 'bob', PASSWORD
'********');

-- Modify the user mapping.
gaussdb=# ALTER USER MAPPING FOR bob SERVER my_server OPTIONS (SET PASSWORD '********');

-- Delete the user mapping.
gaussdb=# DROP USER MAPPING FOR bob SERVER my_server;

-- Delete the foreign server.
gaussdb=# DROP SERVER my_server;

-- Delete the role.
gaussdb=# DROP ROLE bob;
```

Helpful Links

ALTER USER MAPPING and **CREATE USER MAPPING**

7.12.9.44 DROP VIEW

Description

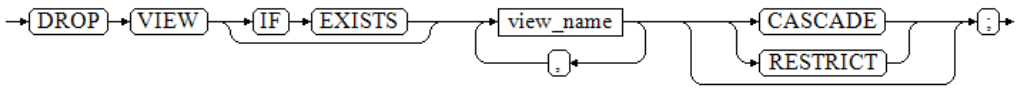
Deletes a view from a database.

Precautions

The owner of a view, owner of the schema of the view, users granted with the DROP permission on the view, or users granted with the DROP ANY TABLE permission can run the **DROP VIEW** command. By default, a system administrator has the permission to run the command when separation of duties is disabled.

Syntax

```
DROP VIEW [ IF EXISTS ] view_name [, ...] [ CASCADE | RESTRICT ];
```



Parameters

- **IF EXISTS**

  Reports a notice instead of an error if the specified view does not exist.
- **view_name**

  Specifies the name of the view to be deleted.

  Value range: an existing view name
- **CASCADE | RESTRICT**
  - **CASCADE**: cascadingly deletes the objects (such as other views) that depend on the view.
  - **RESTRICT**: refuses to delete the view if any objects depend on it. This is the default action.

Examples

See **Examples** in section "CREATE VIEW."

Helpful Links

ALTER VIEW and **CREATE VIEW**

7.12.9.45 DROP WEAK PASSWORD DICTIONARY

Description

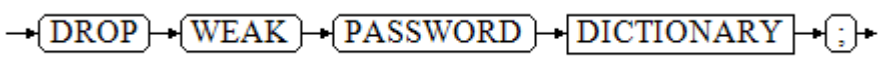
DROP WEAK PASSWORD DICTIONARY is used to clear the weak password dictionary. You can run this statement to clear all weak passwords in the gs_global_config system catalog.

Precautions

Only the initial user, system administrator, and security administrator have the permission to execute this syntax.

Syntax

```
DROP WEAK PASSWORD DICTIONARY;
```



Examples

```
-- Insert a single weak password into the gs_global_config system catalog.
gaussdb=# CREATE WEAK PASSWORD DICTIONARY WITH VALUES ('********');

-- Check weak passwords in the gs_global_config system catalog.
gaussdb=# SELECT * FROM gs_global_config WHERE NAME LIKE 'weak_password';
      name       |    value
-----------------+--------------
 weak_password   | ********
(1 rows)

-- Clear all weak passwords in the gs_global_config system catalog.
gaussdb=# DROP WEAK PASSWORD DICTIONARY;

-- View existing weak passwords.
gaussdb=# SELECT * FROM gs_global_config WHERE NAME LIKE 'weak_password';
 name | value
------+-------
(0 rows)
```

Helpful Links

CREATE WEAK PASSWORD DICTIONARY

7.12.10 E

7.12.10.1 EXECUTE

Description

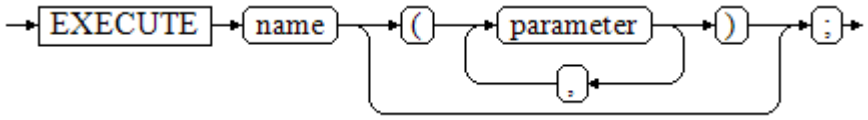
Executes a prepared statement. Because a prepared statement exists only in the lifetime of the session, the prepared statement must be created earlier in the current session by using the PREPARE statement.

Precautions

If the PREPARE statement declares some parameters when the prepared statement is created, the parameter set passed to the EXECUTE statement must be compatible. Otherwise, an error occurs.

Syntax

```
EXECUTE name [ ( parameter [, ...] ) ];
```

Parameters

- **name**

  Specifies the name of the prepared statement to be executed.

- **parameter**

  Specifies a parameter of the prepared statement. It must be an expression that generates a value compatible with the data type of the parameter specified when the prepared statement was created. **ROWNUM** cannot be used as a parameter.

Examples

```
-- Create the reason table.
gaussdb=# CREATE TABLE reason (
    CD_DEMO_SK          int NOT NULL,
    CD_GENDER           varchar(10),
    CD_MARITAL_STATUS   varchar(10)
);

-- Create a prepared statement for an INSERT statement and execute the prepared statement.
gaussdb=# PREPARE insert_reason(int,varchar(10),varchar(10)) AS INSERT INTO reason VALUES($1,$2,$3);
gaussdb=# EXECUTE insert_reason(52, 'AAAAAAAADD', 'reason 52');

-- Query data.
gaussdb=# SELECT * FROM reason;
 cd_demo_sk | cd_gender  | cd_marital_status
------------+------------+-------------------
         52 | AAAAAAAADD | reason 52
(1 row)

-- Delete the reason table.
gaussdb=# DROP TABLE reason;
```

Helpful Links

7.13.16.2-PREPARE and **DEALLOCATE**

7.12.10.2 EXPDP DATABASE

Function

EXPDP DATABASE exports all physical files in a database.

Syntax

```
EXPDP DATABASE db_name LOCATION = 'directory';
```

Parameter Description

- **db_name**

  Name of the database to be exported.

- **directory**

  Directory for storing exported files.

Examples

```
-- The EXPDP DATABASE syntax is used for fine-grained backup and restoration and is called by the backup and restoration tool. If you directly call the syntax, an error message may be displayed, indicating that the directory does not exist. Therefore, you are not advised to directly call the syntax.
gaussdb=# EXPDP DATABASE test LOCATION = '/data1/expdp/database';
```

7.12.10.3 EXPDP TABLE

Function

EXPDP TABLE exports all table-related index, sequence, partition, toast, and toast index files.

Syntax

```
EXPDP TABLE table_name LOCATION = 'directory';
```

Parameter Description

- **table_name**

  Name of the table to be exported.

- **directory**

  Directory for storing exported files.

Examples

```
-- The EXPDP TABLE syntax is used for fine-grained backup and restoration and is called by the backup and restoration tool. If you directly call the syntax, an error message may be displayed, indicating that the directory does not exist. Therefore, you are not advised to directly call the syntax.
gaussdb=# EXPDP TABLE test_t LOCATION = '/data1/expdp/table0';
```

7.12.10.4 EXPLAIN

Description

Shows the execution plan of an SQL statement.

The execution plan shows how the tables referenced by the statement will be scanned - by plain sequential scan, index scan, etc. - and if multiple tables are referenced, what join algorithms will be used to bring together the required rows from each input table.

The most critical part of the display is the estimated statement execution cost, which is the planner's guess at how long it will take to run the statement.

The **ANALYZE** option causes the statement to be actually executed, not only planned. The total elapsed time expended within each plan node (in milliseconds) and total number of rows it actually returned are added to the display. This is useful for determining whether the plan generator's estimate is close to the actual value.

Precautions

The statement is actually executed when ANALYZE is used. If you want to use EXPLAIN ANALYZE on an INSERT, UPDATE, DELETE, CREATE TABLE AS, or EXECUTE statement without letting the statement affect your data, use the following approach:

```
START TRANSACTION;
EXPLAIN ANALYZE ...;
ROLLBACK;
```

Syntax

- Display the execution plan of an SQL statement, which supports multiple options and has no requirements for the order of options.

  ```
  EXPLAIN [ (  option  [, ...] )  ] statement;
  ```

  The syntax of the **option** clause is as follows:

  ```
  ANALYZE [ boolean ] |
     ANALYSE [ boolean ] |
     VERBOSE [ boolean ] |
     COSTS [ boolean ] |
     CPU [ boolean ] |
     DETAIL [ boolean ] |
     BUFFERS [ boolean ] |
     TIMING [ boolean ] |
     PLAN [ boolean ] |
     BLOCKNAME [ boolean ] |
     OUTLINE [ boolean ] |
     ADAPTCOST [ boolean ] |
     FORMAT { TEXT | XML | JSON | YAML }
     OPTEVAL [ boolean ]
  ```

- Display the execution plan of an SQL statement, where options are in order.

  ```
  EXPLAIN  { [  ANALYZE  | ANALYSE  ] [ VERBOSE  ]  | PERFORMANCE  } statement;
  ```

Parameters

- **statement**

  Specifies the SQL statement to explain.

- **ANALYZE boolean | ANALYSE boolean**

  Specifies whether to display actual run times and other statistics. When two parameters are used at the same time, the latter one in **option** takes effect.

  Value range:

  - **TRUE** (default): displays them.
  - **FALSE**: does not display them.

- **VERBOSE boolean**

  Specifies whether to display additional information regarding the plan.

  Value range:

  - **TRUE** (default): displays it.
  - **FALSE**: does not display it.

- **COSTS boolean**

  Specifies whether to display the estimated total cost of each plan node, estimated number of rows, estimated width of each row.

  Value range:

  - **TRUE** (default): displays them.
  - **FALSE**: does not display them.
- **CPU boolean**

  Specifies whether to display CPU usage. This parameter must be used together with ANALYZE or ANALYSE.

  Value range:
  - **TRUE** (default): displays it.
  - **FALSE**: does not display it.
- **DETAIL boolean**

  Displays information about database nodes. This parameter must be used together with ANALYZE or ANALYSE.

  Value range:
  - **TRUE** (default): displays it.
  - **FALSE**: does not display it.
- **BUFFERS boolean**

  Specifies whether to display buffer usage. This parameter must be used together with ANALYZE or ANALYSE.

  Value range:
  - **TRUE**: displays it.
  - **FALSE** (default): does not display it.
- **TIMING boolean**

  Specifies whether to display the actual startup time and time spent on the output node. This parameter must be used together with ANALYZE or ANALYSE.

  Value range:
  - **TRUE** (default): displays them.
  - **FALSE**: does not display them.
- **PLAN boolean**

  Specifies whether to store the execution plan in **PLAN_TABLE**. If this parameter is set to **on**, the execution plan is stored in **PLAN_TABLE** and not displayed on the screen. Therefore, this parameter cannot be used together with other parameters when it is set to **on**.

  Value range:
  - **ON** (default): The execution plan is stored in **PLAN_TABLE** and not printed on the screen. If the plan is stored successfully, "EXPLAIN SUCCESS" is returned.
  - **OFF**: The execution plan is not stored but is printed on the screen.
- **BLOCKNAME boolean**

  Specifies whether to display the query block where each operation of the plan is located. When this option is enabled, the name of the query block where each operation is performed is displayed in the **Query Block** column. This helps users obtain the query block name and use hints to modify the execution plan.
  - **TRUE** (default value): When the plan is displayed, the name of the query block where each operation is located is displayed in the **Query Block**

column. This option must be used in the pretty mode. See **Hint for Specifying the Query Block Where the Hint Is Located**.

  - **FALSE**: The plan display is not affected.

- **OUTLINE boolean**

  Determines whether to display outline hints of a plan.

  - **ON**: When a plan is displayed, outline hints are displayed below the plan. This option must be used in the pretty mode. See **Outline Hint**.
  - **OFF** (default value): The outline hints of the plan are not displayed.

- **ADAPTCOST boolean**

  Specifies whether to display the cardinality estimation method of a plan in the normal mode.

  - **ON** (default): Display the cardinality estimation method, including the default method and feedback method, on the plan node in the normal mode. This parameter does not take effect for prepared statements.
  - **OFF**: Do not display the cardinality estimation method.

- **FORMAT**

  Specifies the output format.

  Value range: **TEXT**, **XML**, **JSON**, and **YAML**

  Default value: **TEXT**

- **PERFORMANCE**

  Prints all relevant information in execution. Some information is described as follows:

  - **ex c/r**: indicates the average number of CPU cycles used by each row, which is equal to **(ex cyc) / (ex row)**.
  - **ex row**: indicates the number of executed rows.
  - **ex cyc**: indicates the number of used CPU cycles.
  - **inc cyc**: indicates the total number of CPU cycles used by subnodes.
  - **shared hit**: indicates the shared buffer hits of the operator.
  - **loops**: indicates the number of operator loop execution times.
  - **total_calls**: indicates the total number of generated elements.
  - **remote query poll time stream gather**: indicates the operator used to listen to the network poll time when data on each DN reaches the CN.
  - **deserialize time**: indicates the time required for deserialization.
  - **estimated time**: indicates the estimated time.

- **OPTEVAL boolean**

  Specifies whether to display the cost elimination details of the scan operator (currently, only seqscan, indexscan, indexonlyscan and bitmapheapscan are supported). When this function is enabled, a plan block named Cost Evaluation Info (identified by plan id) is displayed in the execution plan. This option can only coexist with COSTS, VERBOSE, and FORMAT. For details about the parameters in the plan block, see **Example 2**.

  Value range:

  - **TRUE**: displays the cost elimination details of the SCAN operator.

- **FALSE** (default): does not display it.

Example 1

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create the tpcds.customer_address table.
gaussdb=# CREATE TABLE tpcds.customer_address
(
ca_address_sk         INTEGER           NOT NULL,
ca_address_id         CHARACTER(16)     NOT NULL
);

-- Insert multiple records into the table.
gaussdb=# INSERT INTO tpcds.customer_address VALUES (5000, 'AAAAAAAABAAAAAAA'),(10000,
'AAAAAAAACAAAAAAA');

-- Create the tpcds.customer_address_p1 table.
gaussdb=# CREATE TABLE tpcds.customer_address_p1 AS TABLE tpcds.customer_address;

-- Change the value of explain_perf_mode to normal.
gaussdb=# SET explain_perf_mode=normal;

-- Display an execution plan for simple queries in the table.
gaussdb=# EXPLAIN SELECT * FROM tpcds.customer_address_p1;
                           QUERY PLAN
-------------------------------------------------------------------------
 Seq Scan on customer_address_p1  (cost=0.00..25.13 rows=1513 width=24)
(1 row)

-- Use the ANALYZE option to add runtime statistics to the output.
gaussdb=# EXPLAIN ANALYZE SELECT * FROM tpcds.customer_address_p1;
                                        QUERY PLAN
------------------------------------------------------------------------------------------------------- Seq Scan on
customer_address_p1  (cost=0.00..25.13 rows=1513 width=24) (actual time=0.015..0.016 rows=2 loops=1)
 Total runtime: 0.108 ms
(2 rows)

-- Use the ANALYZE and CPU options to output the CPU usage information.
gaussdb=# EXPLAIN (ANALYZE,CPU)SELECT * FROM tpcds.customer_address_p1;
                                        QUERY PLAN
------------------------------------------------------------------------------------------------------- Seq Scan on
customer_address_p1  (cost=0.00..25.13 rows=1513 width=24) (actual time=0.011..0.012 rows=2 loops=1)
   (CPU: ex c/r=47397856228403856, ex row=2, ex cyc=94795712456807712, inc cyc=94795712456807712)
 Total runtime: 0.092 ms
(3 rows)

-- Generate an execution plan in JSON format (with explain_perf_mode being normal).
gaussdb=# EXPLAIN(FORMAT JSON) SELECT * FROM tpcds.customer_address_p1;
                QUERY PLAN
-----------------------------------------------
 [                                     +
  {                                    +
    "Plan": {                            +
      "Node Type": "Seq Scan",             +
      "Relation Name": "customer_address_p1",+
      "Alias": "customer_address_p1",        +
      "Startup Cost": 0.00,                +
      "Total Cost": 25.13,                 +
      "Plan Rows": 1513,                    +
      "Plan Width": 24                     +
    }                                  +
  }                                    +
 ]
(1 row)

-- Generate an execution plan in YAML format (with explain_perf_mode being normal).
gaussdb=# EXPLAIN(FORMAT YAML) SELECT * FROM tpcds.customer_address_p1 WHERE
```



```
ca_address_sk=10000;
                QUERY PLAN
------------------------------------------
 - Plan:                              +
     Node Type: "Seq Scan"              +
     Relation Name: "customer_address_p1"+
     Alias: "customer_address_p1"       +
     Startup Cost: 0.00                  +
     Total Cost: 28.91                  +
     Plan Rows: 8                       +
     Plan Width: 24                      +
     Filter: "(ca_address_sk = 10000)"
(1 row)

-- Suppress the execution plan of cost estimation.
gaussdb=# EXPLAIN(COSTS FALSE) SELECT * FROM tpcds.customer_address_p1 WHERE
ca_address_sk=10000;
           QUERY PLAN
---------------------------------
 Seq Scan on customer_address_p1
   Filter: (ca_address_sk = 10000)
(2 rows)


-- Generate an execution plan with aggregate functions for a query.
gaussdb=# EXPLAIN SELECT SUM(ca_address_sk) FROM tpcds.customer_address_p1 WHERE
ca_address_sk<10000;
                            QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate  (cost=30.17..30.18 rows=1 width=12)
   ->  Seq Scan on customer_address_p1  (cost=0.00..28.91 rows=504 width=4)
         Filter: (ca_address_sk < 10000)
(3 rows)

-- Create a level-2 partitioned table.
gaussdb=# CREATE TABLE range_list
(
    month_code VARCHAR2 ( 30 ) NOT NULL ,
    dept_code  VARCHAR2 ( 30 ) NOT NULL ,
    user_no    VARCHAR2 ( 30 ) NOT NULL ,
    sales_amt  int
)
PARTITION BY RANGE (month_code) SUBPARTITION BY LIST (dept_code)
(
  PARTITION p_201901 VALUES LESS THAN( '201903' )
  (
    SUBPARTITION p_201901_a values ('1'),
    SUBPARTITION p_201901_b values ('2')
  ),
  PARTITION p_201902 VALUES LESS THAN( '201910' )
  (
    SUBPARTITION p_201902_a values ('1'),
    SUBPARTITION p_201902_b values ('2')
  )
);

-- Run a query statement containing a level-2 partitioned table.
-- Iterations and Sub Iterations specify the numbers of level-1 and level-2 partitions that are traversed,
respectively.
-- Selected Partitions specifies which level-1 partitions are actually scanned. Selected Subpartitions (p:s)
indicates that s level-2 partitions under the pth level-1 partition are actually scanned. If all level-2 partitions
under the level-1 partition are scanned, the value of s is ALL.
gaussdb=# EXPLAIN SELECT * FROM range_list WHERE dept_code = '1';
                            QUERY PLAN
-------------------------------------------------------------------------------
 Partition Iterator  (cost=0.00..17.68 rows=3 width=103)
   Iterations: 2, Sub Iterations: 2
   ->  Partitioned Seq Scan on range_list  (cost=0.00..17.68 rows=3 width=103)
         Filter: ((dept_code)::text = '1'::text)
```



```
      Selected Partitions:  1..2
      Selected Subpartitions:  1:1  2:1
(6 rows)

-- Delete the tpcds.customer_address_p1 table.
gaussdb=# DROP TABLE tpcds.customer_address_p1;

-- Delete the tpcds.customer_address table.
gaussdb=# DROP TABLE tpcds.customer_address;

-- Delete the range_list table.
gaussdb=# DROP TABLE range_list;

-- Delete a schema.
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

Example 2

```
-- Create tb_a and tb_b.
gaussdb=# CREATE TABLE tb_a(c1 int);
gaussdb=# INSERT INTO tb_a VALUES(1),(2),(3);
gaussdb=# CREATE TABLE tb_b AS SELECT * FROM tb_a;

-- Display the cost elimination details of the SCAN operator.
gaussdb=# EXPLAIN (OPTEVAL on )SELECT * FROM tb_a a, tb_b b WHERE a.c1=b.c1 AND a.c1=1;
                          QUERY PLAN
------------------------------------------------------------------------
 Nested Loop  (cost=0.00..81.88 rows=144 width=8)
   ->  Seq Scan on tb_a a  (cost=0.00..40.03 rows=12 width=4)
         Filter: (c1 = 1)
   ->  Materialize  (cost=0.00..40.09 rows=12 width=4)
         ->  Seq Scan on tb_b b  (cost=0.00..40.03 rows=12 width=4)
               Filter: (c1 = 1)
(6 rows)

-- Delete the tb_a and tb_b tables.
gaussdb=# DROP TABLE tb_a;
gaussdb=# DROP TABLE tb_b;
```


NOTE

For the Cost Evaluation Info (identified by plan id) plan block:

1. The rows **2 --** and **4 --** represent the current winning operator, and the indentation plan blocks under each row represent the operators directly eliminated by the current winning operator. For example, operator 2 eliminates the Seq Scan and Bitmap Heap Scan operators.
2. The preceding key parameters are described as follows:
   1. **id** indicates that the current operator is converted from a path with a specified ID. The value is used to locate a path in debug2 logs.
   2. **rpage** indicates the number of pages in the base table used during the cost model calculation.
   3. **ipage** indicates the number of pages in the index used during the cost model calculation.
   4. **tuples** indicates the number of tuples used in the cost model calculation.
   5. **selec** indicates the index selectivity used during cost model calculation. The value **–1** indicates that the index selectivity of the current operator is invalid.
   6. **ml** indicates the ML model event that is currently triggered during cost model calculation. The value **1** indicates that the cache specified by **effective_cache_size** is sufficient, the value **2** indicates that the cache specified by **effective_cache_size** is insufficient, and the value **3** indicates that the cache specified by **effective_cache_size** is severely insufficient.
   7. **iscost** specifies whether an event of ignoring the startup cost has occurred during model cost calculation.
   8. **lossy** specifies whether to trigger the lossy mechanism of bitmap heap scan during cost model calculation.
   9. **uidx** specifies whether to trigger the unique index first rule during cost model calculation.

Helpful Links

ANALYZE | ANALYSE

7.12.10.5 EXPLAIN PLAN

Description

EXPLAIN PLAN saves information about an execution plan into the **PLAN_TABLE** table. Different from the EXPLAIN statement, EXPLAIN PLAN only saves plan information and does not print information on the screen.

Precautions

- The database node cannot collect plan information about SQL statements that are incorrectly executed.
- Data in **PLAN_TABLE** is in a session-level lifecycle. Sessions are isolated from users, and therefore users can only view the data of the current session and current user.
- **PLAN**: saves plan information into **PLAN_TABLE**. If information is stored successfully, "EXPLAIN SUCCESS" is returned.
- **STATEMENT_ID**: tags each query. The tag information will be stored in **PLAN_TABLE**.

- After the EXPLAIN PLAN statement is executed, plan information is automatically stored in **PLAN_TABLE**. INSERT, UPDATE, and ANALYZE cannot be performed on **PLAN_TABLE**. For details about **PLAN_TABLE**, see **PLAN_TABLE**.

Syntax

```
EXPLAIN PLAN
[ SET STATEMENT_ID = name ]
FOR statement ;
```

Parameters

- **name**

  Specifies a query tag.

  Value range: a string

  NOTE

  If the EXPLAIN PLAN statement does not contain **SET STATEMENT_ID**, **STATEMENT_ID** is empty by default. In addition, the value of **STATEMENT_ID** cannot exceed 30 bytes. Otherwise, an error will be reported.

- **statement**

  Specifies the SQL statement to explain.

Examples

```
-- Create tables foo1 and foo2.
gaussdb=# CREATE TABLE foo1(f1 int, f2 text, f3 text[]);
gaussdb=# CREATE TABLE foo2(f1 int, f2 text, f3 text[]);

-- Run EXPLAIN PLAN.
gaussdb=# EXPLAIN PLAN SET STATEMENT_ID = 'TPCH-Q4' FOR SELECT f1, count(*) FROM foo1 WHERE f1
> 1 AND f1 < 3 AND EXISTS (SELECT * FROM foo2) GROUP BY f1;

-- Query PLAN_TABLE.
gaussdb=# SELECT * FROM plan_table;

-- Delete data from the PLAN_TABLE table and delete tables foo1 and foo2.
gaussdb=# DELETE FROM plan_table WHERE STATEMENT_ID = 'TPCH-Q4';
gaussdb=# DROP TABLE foo1;
gaussdb=# DROP TABLE foo2;
```

7.12.11 F

7.12.11.1 FETCH

Description

FETCH retrieves rows using a previously created cursor.

A cursor has an associated position, which is used by FETCH. The cursor position can be before the first row of the query result, on any particular row of the result, or after the last row of the result.

- When created, a cursor is positioned before the first row.

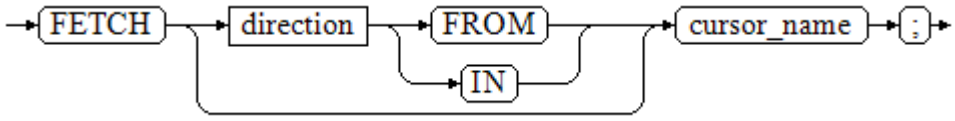
- After fetching some rows, the cursor is positioned on the row most recently retrieved.
- If FETCH runs off the end of the available rows then the cursor is left positioned after the last row, or before the first row if fetching backward.
- FETCH ALL or FETCH BACKWARD ALL will always leave the cursor positioned after the last row or before the first row.

Precautions

- If the cursor is declared with **NO SCROLL**, backward fetches like FETCH BACKWARD are not allowed.
- The forms **NEXT**, **PRIOR**, **FIRST**, **LAST**, **ABSOLUTE**, and **RELATIVE** fetch a single row after moving the cursor appropriately. If there is no such row, an empty result is returned, and the cursor is left positioned before the first row (backward fetch) or after the last row (forward fetch) as appropriate.
- The forms using **FORWARD** and **BACKWARD** retrieve the indicated number of rows moving in the forward or backward direction, leaving the cursor positioned on the last-returned row or after (backward fetch)/before (forward fetch) all rows if the **count** exceeds the number of rows available.
- **RELATIVE 0**, **FORWARD 0**, and **BACKWARD 0** all request fetching the current row without moving the cursor, that is, re-fetching the most recently fetched row. This action will succeed unless the cursor is positioned before the first row or after the last row. If the cursor is positioned before the first row or after the last row, no row is returned.

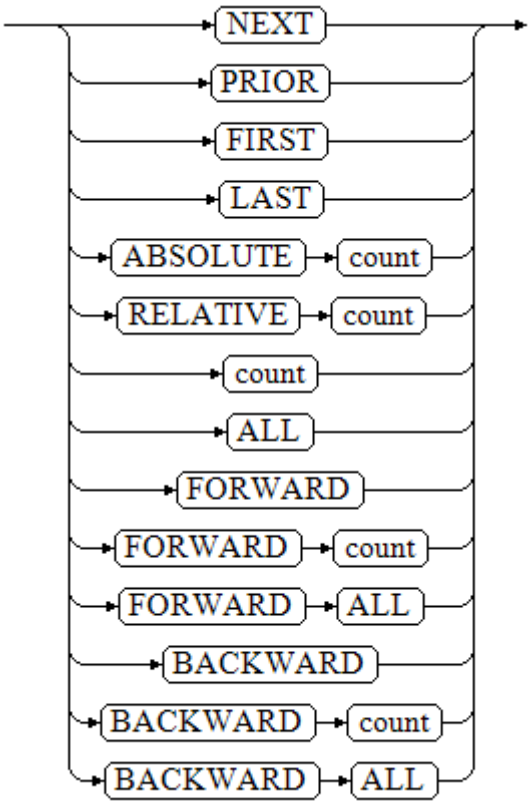
Syntax

```
FETCH [ direction { FROM | IN } ] cursor_name;
```



The **direction** clause specifies optional parameters.

```
NEXT
  | PRIOR
  | FIRST
  | LAST
  | ABSOLUTE count
  | RELATIVE count
  | count
  | ALL
  | FORWARD
  | FORWARD count
  | FORWARD ALL
  | BACKWARD
  | BACKWARD count
  | BACKWARD ALL
```

Parameters

- **direction**

  Defines the fetch direction.

  Value range:

  – NEXT (default value)

  Fetches the next row.

  – PRIOR

  Fetches the prior row.

  – FIRST

  Fetches the first row of the query (same as **ABSOLUTE 1**).

  – LAST

  Fetches the last row of the query (same as **ABSOLUTE -1**).

  – ABSOLUTE count

  Fetches the *count*'th row of the query.

  ABSOLUTE fetches are not any faster than navigating to the desired row with a relative move: the underlying implementation must traverse all the intermediate rows anyway.

  Value range: a possibly-signed integer

  - If *count* is positive, the *count*'th row of the query will be fetched.
  - If *count* is negative, the **abs(***count***)**th row from the end of the query result will be fetched.

  - If *count* is set to **0**, the cursor is positioned before the first row.
- RELATIVE count

  Fetches the *count*'th succeeding row or the *count*'th prior row if count is negative.

  Value range: a possibly-signed integer

  - If *count* is positive, the *count*th succeeding row will be fetched.
  - If **count** is a negative integer, fetches the abs(count)'th prior row.
  - If the current row contains no data, **RELATIVE 0** returns null.
- count

  Fetches the next *count* rows (same as **FORWARD** *count*).
- ALL

  Fetches all remaining rows (same as **FORWARD ALL**).
- FORWARD

  Fetches the next row (same as **NEXT**).
- FORWARD count

  Fetches the *count* succeeding rows or *count* prior rows if *count* is negative.
- FORWARD ALL

  Fetches all remaining rows.
- BACKWARD

  Fetches the prior row (same as **PRIOR**).
- BACKWARD count

  Fetches the prior *count* rows (scanning backwards).

  Value range: a possibly-signed integer

  - If *count* is positive, the prior *count* rows will be fetched.
  - If *count* is negative, the succeeding *abs (count)* rows will be fetched.
  - **BACKWARD 0** re-fetches the current row, if any.
- BACKWARD ALL

  Fetches all prior rows (scanning backwards).

- **{ FROM | IN } cursor_name**

  Specifies the cursor name using the keyword **FROM** or **IN**.

  Value range: an existing cursor name

Examples

```
-- Create the test table and insert 20 data records into the table.
gaussdb=# CREATE TABLE test(c1 int, c2 int);
gaussdb=# INSERT INTO test VALUES (generate_series(1,20),generate_series(1,20));

-- Start a transaction and create a cursor named cursor1.
gaussdb=# START TRANSACTION;
gaussdb=# CURSOR cursor1 FOR SELECT * FROM test ORDER BY 1;
```



```
-- The cursor retrieves three rows of data from the associated position.
gaussdb=# FETCH FORWARD 3 FROM cursor1;
 c1 | c2
----+----
  1 |  1
  2 |  2
  3 |  3
(3 rows)

-- Close the cursor and commit the transaction.
gaussdb=# CLOSE cursor1;
gaussdb=# END;

-- Delete the table.
gaussdb=# DROP TABLE test;
```

Helpful Links

CLOSE, **MOVE**, and **CURSOR**

7.12.12 G

7.12.12.1 GRANT

Description

Grants permissions to roles and users.

GRANT is used in the following scenarios:

- **Granting system permissions to roles or users**

  System permissions are also called user attributes, including SYSADMIN, CREATEDB, CREATEROLE, AUDITADMIN, MONADMIN, OPRADMIN, POLADMIN, INHERIT, REPLICATION, and LOGIN.

  They can be specified only by the CREATE ROLE or ALTER ROLE statement. The SYSADMIN permissions can be granted and revoked using GRANT ALL PRIVILEGE and REVOKE ALL PRIVILEGE, respectively. System permissions cannot be inherited by a user from a role, and cannot be granted using PUBLIC.

- **Granting database object permissions to roles or users**

  Grant permissions on a database object (table, view, column, database, function, schema, or tablespace) to a role or user.

  GRANT gives specific permissions on a database object to one or more roles. These permissions are added to those already granted, if any.

  The keyword PUBLIC indicates that the permissions are to be granted to all roles, including those that might be created later. PUBLIC can be thought of as an implicitly defined group that always includes all roles. Any particular role will have the sum of permissions granted directly to it, permissions granted to any role it is presently a member of, and permissions granted to PUBLIC.

  If WITH GRANT OPTION is specified, the recipient of the permission can in turn grant it to others. Without a grant option, the recipient cannot do that. This option cannot be granted to PUBLIC, which is a unique GaussDB attribute.

GaussDB grants the permissions on certain types of objects to PUBLIC. By default, permissions on tables, columns, sequences, foreign data sources, foreign servers, schemas, and tablespaces are not granted to PUBLIC users, but the following permissions are granted to PUBLIC users: CONNECT and CREATE TEMP TABLE permissions on databases, EXECUTE permission on functions, and USAGE permission on languages and data types (including domains). An object owner can revoke the default permissions granted to PUBLIC users and grant permissions to other users as needed. For security purposes, you are advised to create an object and set its permissions in the same transaction so that other users do not have time windows to use the object. In addition, you can restrict the permissions of the PUBLIC user group by referring to t"Permission Management" in *Security Hardening Guide*. These default permissions can be modified by running the **ALTER DEFAULT PRIVILEGES** command.

By default, an object owner has all permissions on the object. For security purposes, the owner can discard some permissions. However, the ALTER, DROP, COMMENT, INDEX, VACUUM, and re-grantable permissions of the object are inherent permissions implicitly owned by the owner.

- **Granting the permissions of one role or user to others**

  Grant the permissions of one role or user to others. In this case, every role or user can be regarded as a set of one or more database permissions.

  If WITH ADMIN OPTION is specified, the recipients can in turn grant the permissions to other roles or users or revoke the permissions they have granted to other roles or users. If recipients' permissions are changed or revoked later, the grantees' permissions will also change.

  When separation-of-duty is disabled, the system administrator can grant or revoke the permissions of any non-permanent user, O&M administrator, or private user role, and the security administrator can grant or revoke the permissions of any non-system administrator, built-in role, permanent user, O&M administrator, or private user role.

- **Granting ANY permissions to roles or users**

  Grant ANY permissions to a specified role or user. For details about the value range of the ANY permissions, see the syntax. If WITH ADMIN OPTION is specified, the grantee can grant the ANY permissions to or revoke them from other roles or users. The ANY permissions can be inherited by a role but cannot be granted to PUBLIC. An initial user and the system administrator when separation of duties is disabled can grant the ANY permissions to or revoke them from any role or user.

  Currently, the following ANY permissions are supported: CREATE ANY TABLE, ALTER ANY TABLE, DROP ANY TABLE, SELECT ANY TABLE, INSERT ANY TABLE, UPDATE ANY TABLE, DELETE ANY TABLE, CREATE ANY SEQUENCE, CREATE ANY INDEX, CREATE ANY FUNCTION, EXECUTE ANY FUNCTION, CREATE ANY PACKAGE, EXECUTE ANY PACKAGE, CREATE ANY TYPE, ALTER ANY TYPE, DROP ANY TYPE, ALTER ANY SEQUENCE, DROP ANY SEQUENCE, SELECT ANY SEQUENCE, ALTER ANY INDEX, DROP ANY INDEX, CREATE ANY SYNONYM, DROP ANY SYNONYM, CREATE ANY TRIGGER, ALTER ANY TRIGGER, and DROP ANY TRIGGER. For details about the ANY permission scope, see **Table 7-223**.

Precautions

- It is not allowed to grant the ANY permissions to PUBLIC or revoke the ANY permissions from PUBLIC.
- The ANY permissions are database permissions and are valid only for database objects that are granted with the permissions. For example, SELECT ANY TABLE only allows a user to view all user table data in the current database, but the user does not have the permission to view user tables in other databases.
- The ANY permissions and the original permissions do not affect each other.
- If a user is granted with the CREATE ANY TABLE permission, the owner of a table created in a schema with the same name as the user is the owner of the schema. When the user performs other operations on the table, the user needs to be granted with the corresponding operation permission. Similarly, if a user is granted with the CREATE ANY FUNCTION, CREATE ANY PACKAGE, CREATE ANY TYPE, CREATE ANY SEQUENCE, or CREATE ANY INDEX permission, the owner of an object created in a schema with the same name is the owner of the schema. If a user is granted with the CREATE ANY TRIGGER or CREATE ANY SYNONYM permission, the owner of an object created in a schema with the same name is the creator.
- Exercise caution when granting the CREATE ANY FUNCTION or CREATE ANY PACKAGE permission to users to prevent other users from using DEFINER functions or PACKAGE for privilege escalation.
- When GRANT is used to grant a user the permission to use a table, if the permission is not properly used, ALTER may be used to add expressions to the default values and constraints of the table, or indexes may be created to add expressions to INDEX. In this case, the permission may be exploited.
- When GRANT is used to grant the TRIGGER permission, if the permission is not properly used, the WHEN condition may be used to create expressions. When the trigger is triggered, the permission may be exploited.
- When granting permissions to users, pay special attention to the definer permission on functions/packages. The definer permission is executed as the owner of the functions/packages. If the permission is not properly granted (including GRANT ROLE TO ROLE), the permission may be exploited.
- Do not grant object permissions to too many users. You can use roles or PUBLIC based on service requirements.

Syntax

- Grant the table or view access permission to a user or role.

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER | ALTER | DROP
| COMMENT | INDEX | VACUUM } [, ...]
    | ALL [ PRIVILEGES ] }
  ON { [ TABLE ] table_name [, ...]
    | ALL TABLES IN SCHEMA schema_name [, ...] }
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- Grant the column access permission to a user or role.

```
GRANT { {{ SELECT | INSERT | UPDATE | REFERENCES | COMMENT } ( column_name [, ...] )} [, ...]
    | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
  ON [ TABLE ] table_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```


> **NOTE**
>
> If you have the permission to access a table, you have the permission to access all columns in the table by default. To grant only the access permission on a column in a table, you need to revoke the access permission on the table.

Grant the sequence access permission to a specified role or user. The **LARGE** column is optional. The assignment statement does not distinguish whether the sequence is LARGE.

```
GRANT { { SELECT | UPDATE | USAGE | ALTER | DROP | COMMENT } [, ...]
    | ALL [ PRIVILEGES ] }
  ON { [ [ LARGE ] SEQUENCE ] sequence_name [, ...]
    | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- Grant the database access permission to a user or role.

```
GRANT { { CREATE | CONNECT | TEMPORARY | TEMP | ALTER | DROP | COMMENT } [, ...]
    | ALL [ PRIVILEGES ] }
  ON DATABASE database_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- Grant the domain access permission to a user or role.

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
  ON DOMAIN domain_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

> **NOTE**
>
> In the current version, the domain access permission cannot be granted.

- Grant the CMK access permission to a specified user or role.

```
GRANT { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
  ON CLIENT_MASTER_KEY client_master_key [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- Grant the column encryption key (CEK) access permission to a specified user or role.

```
GRANT { { USAGE | DROP } [, ...] | ALL [ PRIVILEGES ] }
  ON COLUMN_ENCRYPTION_KEY column_encryption_key [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- Grant the foreign data source access permission to a user or role.

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
  ON FOREIGN DATA WRAPPER fdw_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- Grant the foreign server access permission to a user or role.

```
GRANT { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
  ON FOREIGN SERVER server_name [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- Grant the function access permission to a user or role.

```
GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
  ON { FUNCTION {function_name ( [ {[ argmode ] [ arg_name ] arg_type} [, ...] ] )} [, ...]
    | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```

- Grant the procedural procedure access permission to a user or role.

```
GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
  ON PROCEDURE {proc_name ( [ {[ argmode ] [ arg_name ] arg_type} [, ...] ] )} [, ...]
  TO { [ GROUP ] role_name | PUBLIC } [, ...]
  [ WITH GRANT OPTION ];
```


- Grant the procedural language access permission to a user or role.

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON LANGUAGE lang_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- Grant the large object access permission to a specified user or role.

```
GRANT { { SELECT | UPDATE } [, ...] | ALL [ PRIVILEGES ] }
    ON LARGE OBJECT loid [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

  NOTE

  In the current version, the large object access permission cannot be granted.

- Grant the schema access permission to a user or role.

```
GRANT { { CREATE | USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON SCHEMA schema_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

  NOTE

  When you grant table or view permissions to other users, you also need to grant the USAGE permission on the schema that the tables and views belong to. Without the USAGE permission, the users with table or view permissions can only see the object names, but cannot access them. This syntax cannot be used to grant the permission to create tables in schemas with the same name, but you can use the syntax for granting permission of a role to another user or role to achieve the same effect.

- Grant the tablespace access permission to a user or role.

```
GRANT { { CREATE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON TABLESPACE tablespace_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

- Grant the type access permission to a user or role.

```
GRANT { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON TYPE type_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...]
    [ WITH GRANT OPTION ];
```

  NOTE

  In the current version, the type access permission cannot be granted.

- Grant the DIRECTORY permission to a role.

```
GRANT { { READ | WRITE | ALTER | DROP } [, ...] | ALL [PRIVILEGES] }
  ON DIRECTORY directory_name [, ...]
  TO { [GROUP] role_name | PUBLIC } [, ...]
  [WITH GRANT OPTION];
```

- Grant the PACKAGE permission to a role.

```
GRANT { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [PRIVILEGES] }
  ON { PACKAGE package_name [, ...]
    | ALL PACKAGES IN SCHEMA schema_name [, ...] }
  TO { [GROUP] role_name | PUBLIC } [, ...]
  [WITH GRANT OPTION];
```

- Grant a role's permissions to another user or role.

```
GRANT role_name [, ...]
  TO role_name [, ...]
  [ WITH ADMIN OPTION ];
```

- Grant the SYSADMIN permission to a role.

```
GRANT ALL { PRIVILEGES | PRIVILEGE }
  TO role_name;
```

- Grant the ANY permissions to another user or role.


```
GRANT { CREATE ANY TABLE | ALTER ANY TABLE | DROP ANY TABLE | SELECT ANY TABLE | INSERT
ANY TABLE | UPDATE ANY TABLE |
  DELETE ANY TABLE | CREATE ANY SEQUENCE | CREATE ANY INDEX | CREATE ANY FUNCTION |
EXECUTE ANY FUNCTION |
  CREATE ANY PACKAGE | EXECUTE ANY PACKAGE | CREATE ANY TYPE | ALTER ANY TYPE | DROP ANY
TYPE | ALTER ANY SEQUENCE | DROP ANY SEQUENCE |
  SELECT ANY SEQUENCE | ALTER ANY INDEX | DROP ANY INDEX | CREATE ANY SYNONYM | DROP
ANY SYNONYM | CREATE ANY TRIGGER | ALTER ANY TRIGGER | DROP ANY TRIGGER
 } [, ...]
  TO [ GROUP ] role_name [, ...]
  [ WITH ADMIN OPTION ];
```

- Grant the database link object permission to a specified user.

  ```
  GRANT { CREATE | ALTER | DROP } [PUBLIC] DATABASE LINK TO role_name;
  ```

  NOTE

  - **PUBLIC**: creates a public database link visible to all users. If this clause is omitted, the database link is private and used only as a compatible API. The data that can be accessed on the remote database depends on the identity used by the database link during connection.
  - When the permission to create a database link is granted to a user, the user can remotely access a database by using the IP address of the remote database. Exercise caution when granting this permission to users.
  - In addition to the statement for directly granting the database link permission, you can also obtain the database link permission by inheriting permission and granting permission to an administrator.
  - For details about database links, see **DATABASE LINK**.

- Grant the permission to create PUBLIC synonyms to a specified user.

  ```
  GRANT { CREATE | DROP } PUBLIC SYNONYM TO role_name [ WITH GRANT OPTION ];
  ```

  Built-in roles (gs_role_public_synonym_create and gs_role_public_synonym_drop) can also be used to grant users the permission to create and delete PUBLIC synonyms.

  - Grant the permission to create PUBLIC synonyms to a user.

    ```
    GRANT gs_role_public_synonym_create TO role_name;
    ```

  - Grant the permission to delete PUBLIC synonyms to a user.

    ```
    GRANT gs_role_public_synonym_drop TO role_name;
    ```

Parameters

GRANT permissions are classified as follows:

- **SELECT**

  Allows SELECT from any column, or the specific columns listed, of the specified table, view, or sequence. The SELECT permission on the corresponding column is also required for UPDATE or DELETE.

- **INSERT**

  Allows INSERT of a new row into a table.

- **UPDATE**

  Allows you to run the **UPDATE** command on any column in the specified table. The **UPDATE** command also requires the SELECT permission to query which rows need to be updated. **SELECT ... FOR UPDATE** and **SELECT ... FOR SHARE** also require this permission on at least one column, in addition to the **SELECT** permission.

- **DELETE**

Allows DELETE of a row from a table. Generally, DELETE also requires the SELECT permission to query which rows need to be deleted.

- **TRUNCATE**

  Allows TRUNCATE on a table.

- **REFERENCES**

  Allows creation of a foreign key constraint referencing a table. This permission is required on both referencing and referenced tables.

- **TRIGGER**

  Allows the creation of a trigger on the specified table.

- **CREATE**
  - For databases, allows new schemas to be created within the database.
  - For schemas, allows new objects to be created within the schema. To rename an existing object, you must own the object and have the CREATE permission on the schema of the object.
  - For tablespaces, allows tables to be created within the tablespace, and allows databases and schemas to be created that have the tablespace as their default tablespace.

- **CONNECT**

  Allows the grantee to connect to the database.

- **EXECUTE**

  Allows calling a function, including use of any operators that are implemented on top of the function.

- **USAGE**
  - For procedural languages, allows use of the language for the creation of functions in that language.
  - For schemas, allows access to objects contained in the schema. Without this permission, it is still possible to see the object names.
  - For sequences, allows use of the **nextval** function.

- **ALTER**

  Allows users to modify the attributes of a specified object, excluding the owner and schema of the object.

- **DROP**

  Allows users to delete specified objects.

- **COMMENT**

  Allows users to define or modify comments of a specified object.

- **INDEX**

  Allows users to create indexes on specified tables, manage indexes on the tables, and perform REINDEX and CLUSTER operations on the tables.

- **VACUUM**

  Allows users to perform ANALYZE and VACUUM operations on specified tables.

- **ALL PRIVILEGES**

  Grants all available permissions to a user or role at a time. Only a system administrator has the GRANT ALL PRIVILEGES permission.

GRANT parameters are as follows:

- **role_name**
  Specifies the username.
- **table_name**
  Specifies the table name.
- **column_name**
  Specifies the column name.
- **schema_name**
  Specifies the schema name.
- **database_name**
  Specifies the database name.
- **function_name**
  Specifies the function name.
- **procedure_name**
  Specifies the stored procedure name.
- **sequence_name**
  Specifies the sequence name.
- **domain_name**
  Specifies the domain type name.
- **fdw_name**
  Specifies the foreign data wrapper name.
- **lang_name**
  Specifies the language name.
- **type_name**
  Specifies the type name.
- **argmode**
  Specifies the parameter mode.
  Value range: a string. It must comply with the **naming convention**.
- **arg_name**
  Specifies the parameter name.
  Value range: a string. It must comply with the **naming convention**.
- **arg_type**
  Specifies the parameter type.
  Value range: a string. It must comply with the **naming convention**.
- **loid**
  Specifies the identifier of the large object that includes this page.
  Value range: a string. It must comply with the **naming convention**.
- **tablespace_name**
  Specifies the tablespace name.
- **client_master_key**

Name of the CMK.

Value range: a string. It must comply with the **naming convention**.

- **column_encryption_key**

  Name of the column encryption key.

  Value range: a string. It must comply with the **naming convention**.

- **directory_name**

  Specifies the directory name.

  Value range: a string. It must comply with the **naming convention**.

- **WITH GRANT OPTION**

  If WITH GRANT OPTION is specified, the recipient of the permission can in turn grant it to others. Without a grant option, the recipient cannot do that. Grant options cannot be granted to PUBLIC.

When a non-owner of an object attempts to grant permissions on the object:

- The statement will fail outright if the user has no permissions whatsoever on the object.
- As long as some permission is available, the statement will proceed, but it will grant only those permissions for which the user has grant options.
- The GRANT ALL PRIVILEGES forms will issue a warning message if no grant options are held, while the other forms will issue a warning if grant options for any of the permissions specifically named in the statement are not held.

NOTE

When separation of duties is disabled, database administrators can access all objects, regardless of object permission settings. This is comparable to the permissions of **root** in a Unix system. As with **root**, it is unwise to operate as a system administrator except when necessary.

- **WITH ADMIN OPTION**

  If WITH ADMIN OPTION is specified for a role, the grantee can grant the role to other roles or users or revoke the role from other roles or users.

  For the ANY permissions, if WITH ADMIN OPTION is specified, the grantee can grant the ANY permissions to or revoke them from other roles or users.

Table 7-223 ANY permissions

System Permission	Description
CREATE ANY TABLE	Users can create tables or views in the public and user schemas. The users must be granted with the permission to create sequences to create a table that contains serial columns.
ALTER ANY TABLE	Users' ALTER permission on tables or views in the public and user schemas. If the users want to modify the unique index of a table to add a primary key constraint or unique constraint to the table, the users must be granted with the index permission on the table.

System Permission	Description
DROP ANY TABLE	Users' DROP permission on tables or views in the public and user schemas.
SELECT ANY TABLE	Users' SELECT permission on tables or views in the public and user schemas, which is still subject to row-level security.
UPDATE ANY TABLE	Users' UPDATE permission on tables or views in the public and user schemas, which is still subject to row-level security.
INSERT ANY TABLE	Users' INSERT permission on tables or views in the public and user schemas.
DELETE ANY TABLE	Users' DELETE permission on tables or views in the public and user schemas, which is still subject to row-level security.
CREATE ANY FUNCTION	Users can create functions or stored procedures in the user schemas.
EXECUTE ANY FUNCTION	Users' EXECUTE permission on functions or stored procedures in the public and user schemas.
CREATE ANY PACKAGE	Users' permission to create packages in the public and user schemas.
EXECUTE ANY PACKAGE	Users' EXECUTE permission on packages in the public and user schemas.
CREATE ANY TYPE	Users can create types in the public and user schemas.
CREATE ANY SEQUENCE	Users can create sequences in the public and user schemas.
CREATE ANY INDEX	Users can create indexes in the public and user schemas. The users must be granted with the permission to create tablespaces to create a partitioned table index in a tablespace.
ALTER ANY TYPE	Users have the ALTER permission on types in public and user schemas, excluding modifying the owner of a type or modifying the schema of a type.
DROP ANY TYPE	Users' DROP permission on types in the public and user schemas.
ALTER ANY SEQUENCE	Users have the ALTER permission on sequences in public and user schemas, excluding modifying the owner of a sequence.
DROP ANY SEQUENCE	Users' DROP permission on sequences in the public and user schemas.
SELECT ANY SEQUENCE	Users' SELECT, USAGE, and UPDATE permissions on sequences in the public and user schemas.

System Permission	Description
ALTER ANY INDEX	Users' ALTER permission on indexes in the public and user schemas. To rename an index, users also need the permission to create objects in the schema where the index is located. If tablespace operations are involved, users need to have the corresponding operation permission on the tablespace. To set an index to **UNUSABLE**, users must have the DROP ANY INDEX permission.
DROP ANY INDEX	Users' DROP permission on indexes in the public and user schemas.
CREATE ANY TRIGGER	Users can create triggers in the public and user schemas.
ALTER ANY TRIGGER	Users' ALTER permission on triggers in the public and user schemas.
DROP ANY TRIGGER	Users' DROP permission on triggers in the public and user schemas.
CREATE ANY SYNONYM	Users can create synonyms in user schema.
DROP ANY SYNONYM	Users' DROP permission on synonyms in the public and user schemas.

NOTE

If a user is granted with any ANY permission, the user has the USAGE permission on the public and user schemas but does not have the USAGE permission on the system schemas except public listed in **Table 13-1**.

Examples

Example: Granting system permissions to a user or role

Create the **joe** user and grant the sysadmin permission to it.

```
gaussdb=# CREATE USER joe PASSWORD '********';
gaussdb=# GRANT ALL PRIVILEGES TO joe;
```

Then **joe** has the sysadmin permission.

Example: Granting object permissions to a user or role

1. Revoke the sysadmin permission from the **joe** user. Grant the usage permission of the **tpcds** schema and all permissions on the **tpcds.reason** table to **joe**.

```
gaussdb=# CREATE SCHEMA tpcds;
CREATE SCHEMA
gaussdb=# CREATE TABLE tpcds.reason
(
r_reason_sk         INTEGER      NOT NULL,
r_reason_id         CHAR(16)     NOT NULL,
```



```
r_reason_desc       VARCHAR(20)
);
CREATE TABLE
gaussdb=# REVOKE ALL PRIVILEGES FROM joe;
gaussdb=# GRANT USAGE ON SCHEMA tpcds TO joe;
gaussdb=# GRANT ALL PRIVILEGES ON tpcds.reason TO joe;
```

Then **joe** has all permissions on the **tpcds.reason** table, including create, retrieve, update, and delete.

2. Grant the retrieve permission of **r_reason_sk**, **r_reason_id**, and **r_reason_desc** columns and the update permission of the **r_reason_desc** column in the **tpcds.reason** table to **joe**.

```
gaussdb=# GRANT SELECT (r_reason_sk,r_reason_id,r_reason_desc),UPDATE (r_reason_desc) ON
tpcds.reason TO joe;
```

Then **joe** has the retrieve permission on the **r_reason_sk**, **r_reason_id**, and **r_reason_desc** columns in the **tpcds.reason** table. To enable **joe** to grant these permissions to other users, execute the following statement:

```
gaussdb=# GRANT SELECT (r_reason_sk, r_reason_id) ON tpcds.reason TO joe WITH GRANT OPTION;
```

Grant the database connection permission and the permission to create schemas in GaussDB to user **joe**, and allow user **joe** to grant this permission to other users.

```
gaussdb=# CREATE DATABASE testdb;
gaussdb=# GRANT CREATE,CONNECT ON DATABASE testdb TO joe WITH GRANT OPTION;
```

Create the **tpcds_manager** role, grant the access and object creation permissions of the **tpcds** schema to **tpcds_manager**, but do not allow **tpcds_manager** to grant these permissions to others.

```
gaussdb=# CREATE ROLE tpcds_manager PASSWORD '********';
gaussdb=# GRANT USAGE,CREATE ON SCHEMA tpcds TO tpcds_manager;
```

Grant all permissions on the **tpcds_tbspc** tablespace to **joe**, but do not allow **joe** to grant these permissions to others.

```
gaussdb=# CREATE TABLESPACE tpcds_tbspc RELATIVE LOCATION 'tablespace/tablespace_1';
gaussdb=# GRANT ALL ON TABLESPACE tpcds_tbspc TO joe;
```

3. Create the fun1 function in the tpcds schema, and grants the ALTER permission of the fun1 function to **joe**.

```
gaussdb=# CREATE or replace FUNCTION tpcds.fun1() RETURN boolean AS
BEGIN
SELECT current_user;
RETURN true;
END;
/
gaussdb=# GRANT ALTER ON FUNCTION tpcds.fun1() TO joe;
```

Example: Granting the permissions of one user or role to others

1. Create the **manager** role, grant **joe**'s permissions to **manager**, and allow **manager** to grant these permissions to others.

```
gaussdb=# CREATE ROLE manager PASSWORD '********';
gaussdb=# GRANT joe TO manager WITH ADMIN OPTION;
```

2. Create the **senior_manager** user and grant **manager**'s permissions to it.

```
gaussdb=# CREATE ROLE senior_manager PASSWORD '********';
gaussdb=# GRANT manager TO senior_manager;
```

3. Revoke permissions and delete users.

```
gaussdb=# REVOKE joe FROM manager;
gaussdb=# REVOKE manager FROM senior_manager;
gaussdb=# DROP USER manager;
gaussdb=# DROP DATABASE testdb;
```

Example: Granting the CMK or CEK permission to other user or role

1. Connect to an encrypted database.

```
-- Use the -C parameter to enable the encrypted database function.
gsql -p 57101 gaussdb -r -C
gaussdb=#  CREATE CLIENT MASTER KEY MyCMK1 WITH ( KEY_STORE = xxx, KEY_PATH = xxx,
ALGORITHM = AES_256_CBC);
CREATE CLIENT MASTER KEY
gaussdb=# CREATE COLUMN ENCRYPTION KEY MyCEK1 WITH VALUES (CLIENT_MASTER_KEY =
MyCMK1, ALGORITHM = AEAD_AES_256_CBC_HMAC_SHA256);
CREATE COLUMN ENCRYPTION KEY
```

2. Create a role **newuser** and grant the key permission to **newuser**.

```
gaussdb=# CREATE USER newuser PASSWORD '********';
CREATE ROLE
gaussdb=# GRANT ALL ON SCHEMA public TO newuser;
GRANT
gaussdb=# GRANT USAGE ON COLUMN_ENCRYPTION_KEY MyCEK1 to newuser;
GRANT
gaussdb=# GRANT USAGE ON CLIENT_MASTER_KEY MyCMK1 to newuser;
GRANT
```

3. Set the user to connect to a database and use a CEK to create an encrypted table.

```
gaussdb=# SET ROLE newuser PASSWORD '********';
gaussdbopenGauss=>  CREATE TABLE acltest1 (x int, x2 varchar(50) ENCRYPTED WITH
(COLUMN_ENCRYPTION_KEY = MyCEK1, ENCRYPTION_TYPE = DETERMINISTIC));
CREATE TABLE
gaussdbopenGauss=> SELECT has_cek_privilege('newuser', 'MyCEK1', 'USAGE');
 has_cek_privilege
-------------------
 t
(1 row)
```

4. Revoke permissions and delete users.

```
gaussdb=# RESET ROLE;
gaussdb=# REVOKE USAGE ON COLUMN_ENCRYPTION_KEY MyCEK1 FROM newuser;
gaussdb=# REVOKE USAGE ON CLIENT_MASTER_KEY MyCMK1 FROM newuser;
gaussdb=# DROP TABLE newuser.acltest1;
gaussdb=# DROP COLUMN ENCRYPTION KEY MyCEK1;
gaussdb=# DROP CLIENT MASTER KEY MyCMK1;
gaussdb=# DROP SCHEMA IF EXISTS newuser CASCADE;
gaussdb=# REVOKE ALL ON SCHEMA public FROM newuser;
gaussdb=# DROP ROLE IF EXISTS newuser;
```

Example: Revoking permissions and deleting roles and users

```
gaussdb=# REVOKE ALTER ON FUNCTION tpcds.fun1() FROM joe;
gaussdb=# REVOKE ALL PRIVILEGES ON tpcds.reason FROM joe;
gaussdb=# REVOKE ALL PRIVILEGES ON SCHEMA tpcds FROM joe;
gaussdb=# REVOKE ALL ON TABLESPACE tpcds_tbspc FROM joe;
gaussdb=# DROP TABLESPACE tpcds_tbspc;
gaussdb=# REVOKE USAGE,CREATE ON SCHEMA tpcds FROM tpcds_manager;
gaussdb=# DROP ROLE tpcds_manager;
gaussdb=# DROP ROLE senior_manager;
gaussdb=# DROP USER joe CASCADE;
gaussdb=# DROP TABLE tpcds.reason;
gaussdb=# DROP FUNCTION tpcds.fun1();
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

Helpful Links

REVOKE and **ALTER DEFAULT PRIVILEGES**

7.12.13 I

7.12.13.1 IMPDP DATABASE CREATE

Function

IMPDP DATABASE CREATE specifies the preparation phase for importing a database.

Syntax

```
IMPDP DATABASE [db_name] CREATE SOURCE = 'directory' OWNER = user [LOCAL];
```

Parameter Description

- **db_name**: name of the new database after the import. If this parameter is not specified, the original database name is retained after the import.
- **directory**: data source directory of the imported database.
- **user**: owner of the imported database.
- **LOCAL**: If this column is specified, data is imported to the original cluster. If this column is not specified, data is imported to a new cluster.

Examples

```
-- The IMPDP DATABASE CREATE syntax is used for fine-grained backup and restoration and is called by the backup and restoration tool. If you directly call the syntax, an error message may be displayed, indicating that the directory does not exist. Therefore, you are not advised to directly call the syntax.
gaussdb=# IMPDP DATABASE test CREATE SOURCE = '/data1/impdp/database' OWNER=admin;
```

7.12.13.2 IMPDP RECOVER

Function

IMPDP RECOVER specifies the execution phase of importing a database.

Syntax

```
IMPDP DATABASE RECOVER SOURCE = 'directory' OWNER = user [LOCAL];
```

Parameter Description

- **directory**: data source directory of the imported database.
- **user**: owner of the imported database.
- **LOCAL**: If this column is specified, data is imported to the original cluster. If this column is not specified, data is imported to a new cluster.

Examples

```
-- The IMPDP DATABASE RECOVER syntax is used for fine-grained backup and restoration and is called by the backup and restoration tool. If you directly call the syntax, an error message may be displayed, indicating that the directory does not exist. Therefore, you are not advised to directly call the syntax.
gaussdb=# IMPDP DATABASE RECOVER SOURCE = '/data1/impdp/database' OWNER=admin;
```


7.12.13.3 IMPDP TABLE

Function

IMPDP TABLE specifies the execution phase of importing a table.

Syntax

```
IMPDP TABLE [AS table_name] SOURCE = 'directory' OWNER = user;
```

Parameter Description

- **table_name**: name of the new table after the import. If this parameter is not specified, the original table name is retained after the import.
- **directory**: data source directory of the imported table.
- **user**: owner of the imported table.

Examples

```
-- The IMPDP TABLE syntax is used for fine-grained backup and restoration and is called by the backup and restoration tool. If you directly call the syntax, an error message may be displayed, indicating that the directory does not exist. Therefore, you are not advised to directly call the syntax.
gaussdb=# IMPDP TABLE SOURCE = '/data1/impdp/table0' OWNER=admin;
```

7.12.13.4 IMPDP TABLE PREPARE

Function

IMPDP TABLE PREPARE specifies the preparation phase for importing a table.

Syntax

```
IMPDP TABLE PREPARE SOURCE = 'directory' OWNER = user;
```

Parameter Description

- **directory**: data source directory of the imported table.
- **user**: owner of the imported table.

Examples

```
-- The IMPDP TABLE PREPARE syntax is used for fine-grained backup and restoration and is called by the backup and restoration tool. If you directly call the syntax, an error message may be displayed, indicating that the directory does not exist. Therefore, you are not advised to directly call the syntax.
gaussdb=# IMPDP TABLE PREPARE SOURCE = '/data1/impdp/table0' OWNER=admin;
```

7.12.13.5 INSERT

Description

Inserts one or more rows of data into a table.

Precautions

- You must have the INSERT permission on a table to insert data to it. If a user is granted with the INSERT ANY TABLE permission, the user has the USAGE

permission on all schemas except system schemas and the INSERT permission on tables in these schemas.

- Use of the RETURNING clause requires the SELECT permission on all columns mentioned in RETURNING.
- If ON DUPLICATE KEY UPDATE is used, you must have the INSERT and UPDATE permissions on the table and the SELECT permission on the columns of the UPDATE clause.
- If you use the query clause to insert rows from a query, you need to have the SELECT permission on any table or column used in the query.
- The generated column cannot be directly written. In the INSERT statement, values cannot be specified for generated columns, but the keyword DEFAULT can be specified.
- When you connect to a database compatible to Teradata and **td_compatible_truncation** is **on**, a long string will be automatically truncated. If later INSERT statements (not involving foreign tables) insert long strings to columns of CHAR- and VARCHAR-typed columns in the target table, the system will truncate the long strings to ensure no strings exceed the maximum length defined in the target table.

  NOTE

  If inserting multi-byte character data (such as Chinese characters) to a database with the character set byte encoding (SQL_ASCII, LATIN1), and the character data crosses the truncation position, the string is truncated based on its bytes instead of characters. Unexpected result will occur in tail after the truncation. If you want correct truncation result, you are advised to adopt encoding set such as UTF8, which has no character data crossing the truncation position.

Syntax

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
INSERT [/*+ plan_hint */] [ IGNORE ] INTO table_name [ { [alias_name] [ ( column_name [, ...] ) ] } |
{ [partition_clause] [ AS alias ] [ ( column_name [, ...] ) ] } ]    { DEFAULT VALUES
   | { VALUES | VALUE } {( { expression | DEFAULT } [, ...] ) }[, ...]
   | query }
   [ ON DUPLICATE KEY UPDATE { NOTHING | { column_name = { expression | DEFAULT } } [, ...] [ WHERE
condition ] }]
   [ RETURNING {* | {output_expression [ [ AS ] output_name ] }[, ...]} ];
```

Insert subqueries and views.

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
INSERT [/*+ plan_hint */] INTO {subquery | view_name} [ AS alias ] [ ( column_name [, ...] ) ]
{ DEFAULT VALUES
   | { VALUES | VALUE } {( { expression | DEFAULT } [, ...] ) }[, ...]
   | query }
   [ RETURNING {* | {output_expression [ [ AS ] output_name ] }[, ...]} ];
```

Parameters

- **WITH [RECURSIVE] with_query [, ...]**

  Specifies one or more subqueries that can be referenced by name in the main query, which is equivalent to a temporary table.

  If **RECURSIVE** is specified, it allows to reference a SELECT subquery by name.

  Format of **with_query**:

  ```
  with_query_name [ ( column_name [, ...] ) ] AS [ [ NOT ] MATERIALIZED ]
  ( {SELECT | VALUES | INSERT | UPDATE | DELETE} )
  ```


– **with_query_name** specifies the name of the result set generated by a subquery. Such names can be used to access the result sets of subqueries in a query.

-- **column_name** specifies the column name displayed in the subquery result set.

– Each subquery can be a SELECT, VALUES, INSERT, UPDATE, or DELETE statement.

NOTICE

If a subquery is a DML statement with a RETURNING statement, the number of command output records of the INSERT statement is determined by the subquery. Assume that table **T1** exists. Run the following statement:

```
WITH CTE AS (INSERT INTO T1 VALUES(1,2) RETURNING *) INSERT INTO T1 SELECT * FROM CTE;
```

The number of command output records of the preceding statement is determined by the following part instead of the entire statement. That is, the command output is "INSERT 0 1" instead of "INSERT 0 2".

```
INSERT INTO T1 VALUES(1,2) RETURNING *
```

– You can use MATERIALIZED or NOT MATERIALIZED to modify the CTE.

– If **MATERIALIZED** is specified, the WITH query will be materialized, and a copy of the subquery result set is generated. The copy is directly queried at the reference point. Therefore, the WITH subquery cannot be jointly optimized with the SELECT statement trunk (for example, predicate pushdown and equivalence class transfer). In this scenario, you can use **NOT MATERIALIZED** for modification. If the WITH query can be executed as a subquery inline, the preceding optimization can be performed.

– If the user does not explicitly declare the materialized attribute, comply with the following rules: If the CTE is referenced only once in the trunk statement to which it belongs and semantically supports inline execution, it will be rewritten as subquery inline execution. Otherwise, the materialized execution will be performed in CTE Scan mode.

NOTE

INSERT ON DUPLICATE KEY UPDATE does not support the WITH or WITH RECURSIVE clauses.

- plan_hint clause

  Follows the INSERT keyword in the **/*+ */** format. It is used to optimize the plan of an INSERT statement block. For details, see **Hint-based Tuning**. In each statement, only the first **/*+** *plan_hint* ***/** comment block takes effect as a hint. Multiple hints can be written.

- **IGNORE**

  When the INSERT statement uses the IGNORE keyword, some ERROR-level errors can be degraded to WARNING-level errors, and invalid values can be adjusted to the closest values based on error scenarios. GaussDB supports the following error codes for error downgrade:

  – Damage to the NOT NULL constraint

  – UNIQUE KEY conflict

- No partition found for the inserted value
- Unmatch between the inserted data and the specified partition
- Multiple rows returned for a subquery
- Oversized data
- Time function overflow
- Division by 0
- Incorrect value

NOTE

1. INSERT IGNORE is supported only when **sql_compatibility** is set to **'B'**, **b_format_version** set to **'5.7'**, and **b_format_dev_version** set to **'s1'**.
2. INSERT IGNORE does not support encrypted tables, foreign tables, and MOTs.
3. INSERT IGNORE does not support PRIMARY KEY constraints or UNIQUE KEY constraints that take effect with a delay.
4. The OIDs of each data type that supports damage to NOT NULL constraints with NULL values are TIMESTAMPOID, TIMESTAMPTZOID, TIMEOID, TIMETZOID, RELTIMEOID, INTERVALOID, TINTERVALOOID, SMALLDATETIMEOID, DATEOID, NAMEOID, POINTOID, PATHOID, POLYGONOID, CIRCLEOID, LSEGOID, BOXOID, JSONOID, JSONBOID, XMLOID, XMLTYPEOID, VARBITOID, NUMERICOID, CIDROID, INETOID, MACADDROID, NUMRANGEOID, INT8RANGEOID, INT4RANGEOID, TSRANGEOID, TSTZRANGEOID, DATERANGEOID, ABSTIMEOID, BOOLOID, INT8OID, INT4OID, INT2OID, INT1OID, FLOAT4OID, FLOAT8OID, CASHOID, UINT1OID, UINT2OID, UINT4OID, UINT8OID, BPCHAROID, VARCHAROID, NVARCHAR2OID, CHAROID, BYTEAOID, RAWOID, BLOBOID, CLOBOID, TEXTOID, YEAROID, and SET. The following table lists the zero values of each data type.
5. When the default zero value is used for calculation, exercise caution when using IGNORE to ignore the NOT NULL constraint.

OID of Each Data Type	Default Zero Value
INT8OID, INT4OID, INT2OID, INT1OID, UINT1OID, UINT2OID, UINT4OID, UINT8OID, FLOAT4OID, FLOAT8OID, NUMERICOID	0 or 0.00 (The number of 0s after the decimal point is specified by a parameter.)
BPCHAROID, VARCHAROID, CHAROID, BYTEAOID, RAWOID, BLOBOID, NVARCHAR2OID, CLOBOID, TEXTOID, VARBITOID, NAMEOID, SET	Empty string
NUMRANGEOID, INT8RANGEOID, INT4RANGEOID, TSRANGEOID, TSTZRANGEOID, DATERANGEOID	empty
TIMEOID	time '00:00:00'
TIMETZOID	timetz '00:00:00'
INTERVALOID	interval '00:00:00'
TINTERVALOOID	tinterval(abstime '1970-01-01 00:00:00', abstime '1970-01-01 00:00:00')

OID of Each Data Type	Default Zero Value
SMALLDATETIMEOID	smalldatetime '1970-01-01 00:00:00'
ABSTIMEOID	abstime '1970-01-01 00:00:00'
RELTIMEOID	reltime '00:00:00'
TIMESTAMPOID, TIMESTAMPTZOID	1970-01-01 00:00:00
DATEOID	1970-01-01
YEAROID	0000
POINTOID	(0,0)
PATHOID, POLYGONOID	((0,0))
CIRCLEOID	<(0,0),0>
LSEGOID	[(0,0),(0,0)]
BOXOID	(0,0),(0,0)
JSONOID, JSONBOID, XMLOID	'null'
XMLTYPEOID	'<null/>'
CIDROID	0.0.0.0/32
INETOID	0.0.0.0
MACADDROID	00:00:00:00:00:00
BOOLOID	f
CASHOID	$0.00

- **table_name**

  Specifies the name of the target table where data will be inserted.

  Value range: an existing table name

  NOTE

  You can use database links to perform operations on remote tables. For details, see **DATABASE LINK**.

- **subquery**

  The inserted object can be a subquery. When a subquery is inserted, the subquery is regarded as a temporary view. The **CHECK OPTION** option can be added to the end of the subquery.

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [/*+ plan_hint */] [ ALL ]
{ * | {expression [ [ AS ] output_name ]} [, ...] }
[ into_option ]
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ [ START WITH condition ] CONNECT BY [NOCYCLE] condition [ ORDER SIBLINGS BY expression ] ]
[ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS
{ FIRST | LAST } ]} [, ...] ]
```



```
[ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
[ into_option ];
```

The specified subquery source **from_item** is as follows:

```
{[ ONLY ] {table_name | view_name} [ * ] [ partition_clause ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
|( select ) [ AS ] alias [ ( column_alias [, ...] ) ]
|with_query_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
|from_item [ NATURAL ] join_type from_item [ ON join_condition | USING ( join_column [, ...] ) ]}
```

If a subquery contains only one table, data is inserted into the table. If a subquery contains multiple tables or has nested relationships, check whether a key-preserved table exists to determine whether data can be inserted. For details about key-preserved tables and **WITH CHECK OPTION**, see **CREATE VIEW**.

- **view_name**

  Specifies the target view to be inserted.

  NOTE

  The restrictions on inserting views and subqueries are as follows:

  1. Only columns that directly reference user columns in the base table can be inserted.
  2. A subquery or view must contain at least one updatable column. For details about updatable columns, see **CREATE VIEW**.
  3. Views and subqueries that contain the DISTINCT, GROUP BY, HAVING, LIMIT, or OFFSET clause at the top layer are not supported.
  4. Views and subqueries that contain set operations (UNION, INTERSECT, EXCEPT, and MINUS) at the top layer are not supported.
  5. Views and subqueries whose target lists contain aggregate functions, window functions, or return set functions (such as array_agg, json_agg, and generate_series) are not supported.
  6. Views with BEFORE/AFTER triggers but without INSTEAD OF triggers or INSTEAD rules are not supported.
  7. The ON DUPLICATE KEY UPDATE function is not supported.
  8. Table types supported in views and subqueries include common tables, temporary tables, global temporary tables, partitioned tables, level-2 partitioned tables, Ustore tables, and Astore tables.
  9. For a multi-table join view or subquery, only one base table can be inserted at a time.
  10. When INSERT joins a view or subquery, explicitly specified columns to be inserted or implicitly specified columns (columns specified during view or subquery creation) cannot reference columns of non-key-preserved tables; if the WITH CHECK OPTION clause is used, INSERT operations cannot be performed on join columns in join views or subqueries. For details about the key-preserved table, see **CREATE VIEW**.
  11. System views cannot be inserted.

- **alias_name**

  Specifies the table alias when the INSERT statement is used without AS alias.

NOTE

1. When the INSERT statement is used without AS alias, the table alias cannot be a keyword (such as SELECT and VALUE) or an expression. The alias must comply with the identifier naming rule.
2. When the INSERT statement is used without AS alias, the table alias cannot be in the INSERT INTO table_name alias_name(alias_name.col1, ...,alias_name.coln) VALUES(xxx); format.
3. When the INSERT statement is used without AS alias, data cannot be inserted to the specified partition.

- **partition_clause**

  Inserts data to a specified partition.

  ```
  PARTITION { ( partition_name ) | FOR ( partition_value [, ...] ) } |
  SUBPARTITION { ( subpartition_name ) | FOR ( subpartition_value [, ...] ) }
  ```

  For details about the keywords, see **SELECT**.

  If the value of the VALUE clause is inconsistent with that of the specified partition, an exception is displayed.

  For details, see **CREATE TABLE SUBPARTITION**.

- **column_name**

  Specifies the name of a column in a table.

  – The column name can be qualified with a subcolumn name or array subscript, if needed.

  – Each column not present in the explicit or implicit column list will be filled with a default value, either its declared default value or **NULL** if there is none. Inserting data into only some columns of a composite type leaves the other columns **NULL**.

  – The target column names **column_name** can be listed in any order. If no list of column names is given at all, the default is all the columns of the table in their declared order.

  – The target columns are the first *N* column names, if there are only *N* columns supplied by the VALUE clause or query.

  – The values provided by the VALUE clause and query are associated with the corresponding columns from left to right in the table.

  Value range: an existing column

- **expression**

  Specifies an expression or a value to assign to the corresponding column.

  – In the INSERT ON DUPLICATE KEY UPDATE statement, expression can be **VALUES(***column_name***)** or **EXCLUDED.***column_name*, indicating that the value of **column_name** corresponding to the conflict row is referenced. **VALUES(***column_name***)** can be nested in a compound expression, for example, **VALUES(***column_name***) + 1**, **VALUES(***column_name***) + VALUES(***column_name***)**, and **function_name(VALUES(***column_name***))**.

  NOTE

  - **VALUES(***column_name***)** can be used only in the ON DUPLICATE KEY UPDATE clause.
  - **VALUES(***column_name***)** cannot be used in the IN or NOT IN expressions.

  – If single-quotation marks are inserted in a column, the single-quotation marks need to be used for escape.

  - If the expression for any column is not of the correct data type, automatic type conversion will be attempted. If the attempt fails, data insertion fails, and the system returns an error message.

- **DEFAULT**

  Specifies the default value of a column. The value is **NULL** if no default value is assigned to it.

- **query**

  Specifies a query statement (SELECT statement) that uses the query result as the inserted data.

- **RETURNING**

  Returns the inserted rows. The syntax of the RETURNING list is identical to that of the output list of SELECT.

- **output_expression**

  Specifies an expression used to calculate the output result of the INSERT statement after each row is inserted.

  Value range: The expression can use any column in the table. You can use the asterisk (*) to return all columns of the inserted row.

- **output_name**

  Specifies a name to use for a returned column.

  Value range: a string. It must comply with the **naming convention**.

- **ON DUPLICATE KEY UPDATE**

  For a table with a unique constraint (UNIQUE INDEX or PRIMARY KEY), if the inserted data violates the unique constraint, the UPDATE clause is executed on the conflicting row to complete the update. For a table without a unique constraint, only the insert operation is performed. If **NOTHING** is specified for UPDATE, this insertion is ignored. You can use **EXCLUDED.** or **VALUES()** to select the column corresponding to the source data.

  - Triggers are supported. The execution sequence of triggers is determined by the actual execution process.
    - Executing INSERT will trigger the BEFORE INSERT and AFTER INSERT triggers.
    - Executing UPDATE will trigger the BEFORE INSERT, BEFORE UPDATE, and AFTER UPDATE triggers.
    - Executing **UPDATE NOTHING** will trigger the BEFORE INSERT trigger.
  - The unique constraint or primary key of DEFERRABLE is not supported.
  - If a table has multiple unique constraints and the inserted data violates multiple unique constraints, only the first row that has a conflict is updated. (The check sequence is closely related to index maintenance. Generally, the conflict check is performed on the index that is created first.)
  - If multiple rows are inserted and these rows conflict with the same row in the table, the system inserts or updates the first row and then updates other rows in sequence.
  - Primary keys and unique index columns cannot be updated.

  - Foreign tables are not supported.
  - The WHERE clause and expression of UPDATE cannot contain sublinks.

Examples

- **Insert a data record.**

  Example:

```
-- Create a table.
gaussdb=# CREATE TABLE test_t1(col1 INT,col2 VARCHAR);

-- Insert data.
gaussdb=# INSERT INTO test_t1 (col1, col2) VALUES (1,'AB');

-- Insert data into some columns of the table.
gaussdb=# INSERT INTO test_t1 (col1) VALUES (2);

-- There is no parenthesis on the left of the VALUES keyword. Values must be added to all fields in the parenthesis on the right according to the table structure sequence.
gaussdb=# INSERT INTO test_t1 VALUES (3,'AC');

-- Query the table.
gaussdb=# SELECT * FROM test_t1;
 col1 | col2
------+------
    1 | AB
    2 |
    3 | AC
(3 rows)

-- Drop the table.
gaussdb=# DROP TABLE test_t1;
```

- **Insert multiple data records.**

  Example:

```
-- Create a table.
gaussdb=# CREATE TABLE test_t2(col1 INT,col2 VARCHAR);
gaussdb=# CREATE TABLE test_t3(col1 INT,col2 VARCHAR);

-- Insert multiple data records.
gaussdb=# INSERT INTO test_t2 (col1, col2) VALUES (10,'AA'),(20,'BB'),(30,'CC');

-- Query the table.
gaussdb=# SELECT * FROM test_t2;
 col1 | col2
------+------
   10 | AA
   20 | BB
   30 | CC
(3 rows)

-- Insert data in test_t2 into test_t3.
gaussdb=# INSERT INTO test_t3 SELECT * FROM test_t2;

-- Query the table.
gaussdb=# SELECT * FROM test_t3;
 col1 | col2
------+------
   10 | AA
   20 | BB
   30 | CC
(3 rows)

-- Drop the table.
gaussdb=# DROP TABLE test_t2;
gaussdb=# DROP TABLE test_t3;
```

- **ON DUPLICATE KEY UPDATE**

Example:

```
-- Create a table.
gaussdb=# CREATE TABLE test_t4 (id INT PRIMARY KEY, info VARCHAR(10));
gaussdb=# INSERT INTO test_t4 VALUES (1, 'AA'), (2,'BB'), (3, 'CC');

-- Use the ON DUPLICATE KEY UPDATE keyword.
gaussdb=# INSERT INTO test_t4 VALUES (3, 'DD'), (4, 'EE') ON DUPLICATE KEY UPDATE info =
VALUES(info);

-- Query the table.
gaussdb=# SELECT * FROM test_t4;
 id | info
----+------
  1 | AA
  2 | BB
  4 | EE
  3 | DD

-- Drop the table.
gaussdb=# DROP TABLE test_t4;
```

- **INSERT IGNORE**

  Example 1: Damage to the NOT NULL constraint

```
-- Create a B-compatible database.
gaussdb=# CREATE DATABASE test DBCOMPATIBILITY ='B';
gaussdb=# \c test
-- Set the pre-installation parameters.
test=# set b_format_version = '5.7';
test=# set b_format_dev_version = 's1';

-- Create a table.
test=# CREATE TABLE test_t5(f1 INT NOT NULL);
CREATE TABLE

-- Use the IGNORE keyword.
test=# INSERT IGNORE INTO test_t5 VALUES(NULL);
WARNING:  null value in column "f1" violates not-null constraint
DETAIL:  Failing row contains (null).
INSERT 0 1

-- Query the table.
test=# SELECT * FROM test_t5;
 f1
----
  0
(1 row)

-- Delete the table.
test=# DROP TABLE test_t5;
```

  Example 2: UNIQUE KEY conflict

```
-- Create a table.
test=# CREATE TABLE test_t6(f1 INT PRIMARY KEY);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "test_t6_pkey" for table "test_t6"
CREATE TABLE

-- Insert data.
test=# INSERT INTO test_t6 VALUES(1);
INSERT 0 1

-- Use the IGNORE keyword.
test=# INSERT IGNORE INTO test_t6 VALUES(1);
WARNING:  duplicate key value violates unique constraint "test_t6_pkey"
INSERT 0 0

-- Query the table.
test=# SELECT * FROM test_t6;
 f1
```



```
----
  1
(1 row)

-- Delete the table.
test=# DROP TABLE test_t6;
DROP TABLE
```

Example 3: No partition found for the inserted value

```
-- Create a table.
test=# CREATE TABLE test_t7(f1 INT, f2 INT) PARTITION BY LIST(f1) (PARTITION p0 VALUES(1, 4, 7),
PARTITION p1 VALUES (2, 5, 8));
CREATE TABLE

-- Use the IGNORE keyword.
test=# INSERT IGNORE INTO test_t7 VALUES(3, 5);
WARNING:  inserted partition key does not map to any table partition
INSERT 0 0

-- Query the table.
test=# SELECT * FROM test_t7;
 f1 | f2
----+----
(0 rows)

-- Delete the table.
test=# DROP TABLE test_t7;
DROP TABLE
```

Example 4: Unmatch between the inserted data and the specified partition

```
-- Create a table.
test=# CREATE TABLE test_t8(f1 INT NOT NULL, f2 TEXT, f3 INT) PARTITION BY RANGE(f1)
(PARTITION p0 VALUES LESS THAN(5), PARTITION p1 VALUES LESS THAN(10), PARTITION p2
VALUES LESS THAN(15), PARTITION p3 VALUES LESS THAN(MAXVALUE));
CREATE TABLE

-- Use the IGNORE keyword.
test=# INSERT IGNORE INTO test_t8 PARTITION(p2) VALUES(20, 'Jan', 1);
WARNING:  inserted partition key does not map to the table partition
DETAIL:  N/A.
INSERT 0 0

-- Query the table.
test=# SELECT * FROM test_t8;
 f1 | f2 | f3
----+----+----
(0 rows)

-- Delete the table.
test=# DROP TABLE test_t8;
DROP TABLE
```

Example 5: Multiple rows returned for a subquery

```
-- Create a table.
test=# CREATE TABLE test_t9(f1 INT, f2 INT);
CREATE TABLE

-- Insert data.
test=# INSERT INTO test_t9 VALUES(1, 1), (2, 2), (3, 3);
INSERT 0 3

-- Use the IGNORE keyword.
test=# INSERT IGNORE INTO test_t9 VALUES((SELECT f1 FROM test_t9), 0);
WARNING:  more than one row returned by a subquery used as an expression
CONTEXT:  referenced column: f1
INSERT 0 1
```



```
-- Query the table.
test=# SELECT * FROM test_t9 WHERE f2 = 0;
 f1 | f2
----+----
    |  0
(1 row)

-- Delete the table.
test=# DROP TABLE test_t9;
DROP TABLE
```

Example 6: Oversized data

```
-- Create a table.
test=# CREATE TABLE test_t10(f1 VARCHAR(5));
CREATE TABLE

-- Use the IGNORE keyword.
test=# INSERT IGNORE INTO test_t10 VALUES('aaaaaaaaa');
WARNING:  value too long for type character varying(5)
CONTEXT:  referenced column: f1
INSERT 0 1

-- Query the table.
test=# SELECT * FROM test_t10;
  f1
-------
 aaaaa
(1 row)

-- Delete the table.
test=# DROP TABLE test_t10;
DROP TABLE
```

Example 7: Time function overflow

```
-- Create a table.
test=# CREATE TABLE test_t11(f1 DATETIME);
CREATE TABLE

-- Use the IGNORE keyword.
test=# INSERT IGNORE INTO test_t11 VALUES(date_sub('2000-01-01', INTERVAL 2001 YEAR));
WARNING:  Datetime function: datetime field overflow
CONTEXT:  referenced column: f1
INSERT 0 1

-- Query the table.
test=# SELECT * FROM test_t11;
 f1
----

(1 row)

-- Delete the table.
test=# DROP TABLE test_t11;
DROP TABLE
```

Example 8: Division by 0

```
-- Create a table.
test=# CREATE TABLE test_t12(f1 INT);
CREATE TABLE

-- Use the IGNORE keyword.
test=# INSERT IGNORE INTO test_t12 VALUES(1/0);
WARNING:  division by zero
CONTEXT:  referenced column: f1
INSERT 0 1

-- Query the table.
test=# SELECT * FROM test_t12;
```



```
 f1
----

(1 row)

-- Delete the table.
test=# DROP TABLE test_t12;
DROP TABLE
```

Example 9: Incorrect value

```
-- Create a table.
test=# CREATE TABLE test_t13(f1 FLOAT);
CREATE TABLE

-- Use the IGNORE keyword.
test=# INSERT IGNORE INTO test_t13 VALUES('1.11aaa');
WARNING:  invalid input syntax for type real: "1.11aaa"
LINE 1: INSERT IGNORE INTO test_t13 VALUES('1.11aaa');
                                           ^
CONTEXT:  referenced column: f1
INSERT 0 1

-- Query the table.
test=# SELECT * FROM  test_t13;
  f1
------
 1.11
(1 row)


-- Delete the table.
test=# DROP TABLE test_t13;
-- Drop a database. (Change the database name based on the actual situation.)
test=# \c test;
test=# DROP DATABASE test;
```

- **WITH [RECURSIVE] with_query [, ...]**

  Example:

```
-- Grade table.
gaussdb=# CREATE TABLE grade (
    sid INT,
    course VARCHAR(20),
    score FLOAT
);

-- Student table.
gaussdb=# CREATE TABLE student(
    sid INT PRIMARY KEY,
    class INT,
    name  VARCHAR(50),
    sex INT CHECK (sex = 0 or sex = 1)
);

-- Insert data.
gaussdb=# WITH student_sid AS ( INSERT INTO student ( sid, CLASS, NAME, sex ) VALUES ( 1, 1,
'Scott', 1 ) RETURNING sid )
    INSERT INTO grade ( sid, course, score )
    VALUE ( ( SELECT sid FROM student_sid ), 'math', '96' ),
      ( ( SELECT sid FROM student_sid ), 'chinese', '82' ),
      ( ( SELECT sid FROM student_sid ), 'english', '86' );

-- Query the table.
gaussdb=# SELECT * FROM student;
 sid | class | name  | sex
-----+-------+-------+-----
   1 |     1 | scott |   1
(1 row)

gaussdb=# SELECT * FROM grade;
```



```
 sid | course  | score
-----+---------+-------
   1 | math    |    96
   1 | chinese |    82
   1 | english |    86
(3 rows)

-- Delete the table.
gaussdb=# DROP TABLE student;
gaussdb=# DROP TABLE grade;
```

- **Insert data into a view or subquery.**

  Example 1: Insert a subquery.

```
-- Create a schema.
gaussdb=# CREATE SCHEMA ins_subqry;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA = 'ins_subqry';
SET

-- Create tables.
gaussdb=# CREATE TABLE t1 (x1 int, y1 int);
CREATE TABLE
gaussdb=# CREATE TABLE t2 (x2 int PRIMARY KEY, y2 int);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "t2_pkey" for table "t2"
CREATE TABLE

-- Insert data into t1 through a subquery.
gaussdb=# INSERT INTO (SELECT * FROM t1) VALUES (1, 1);
INSERT 0 1
gaussdb=# INSERT INTO (SELECT * FROM t1 WHERE y1 < 3) VALUES (5, 5);
INSERT 0 1

-- Insert a subquery with CHECK OPTION specified.
gaussdb=# INSERT INTO (SELECT * FROM t1 WHERE y1 < 3 WITH CHECK OPTION) VALUES (5, 5);
ERROR:  new row violates WITH CHECK OPTION for view "__unnamed_subquery__"
DETAIL:  Failing row contains (5, 5).

-- Insert a subquery with READONLY specified.
gaussdb=# INSERT INTO (SELECT * FROM t1 WITH READ ONLY) VALUES (5, 5);
ERROR:  cannot perform a DML operation on a read-only subquery.

-- Insert a multi-table join subquery.
gaussdb=# INSERT INTO (SELECT * FROM t1, t2 WHERE x1 = x2) (x1, y1) VALUES (2, 2);
INSERT 0 1

-- Delete the schema.
gaussdb=# RESET CURRENT_SCHEMA;
RESET
gaussdb=# DROP SCHEMA ins_subqry CASCADE;
NOTICE:  drop cascades to 2 other objects
DETAIL:  drop cascades to table ins_subqry.t1
drop cascades to table ins_subqry.t2
DROP SCHEMA
```

  Example 2: Insert a view.

```
-- Create a schema.
gaussdb=# CREATE SCHEMA ins_view;
CRETAE SCHEMA
gaussdb=# SET CURRENT_SCHEMA = 'ins_view';
SET

-- Create tables.
gaussdb=# CREATE TABLE t1 (x1 int, y1 int);
CREATE TABLE
gaussdb=# CREATE TABLE t2 (x2 int PRIMARY KEY, y2 int);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "t2_pkey" for table "t2"
CREATE TABLE
```



```
-- Create single-table views.
gaussdb=# CREATE VIEW v_ins1 AS SELECT * FROM t1;
CREATE VIEW
gaussdb=# CREATE VIEW v_ins2 AS SELECT * FROM t1 WHERE y1 < 3;
CREATE VIEW
gaussdb=# CREATE VIEW v_ins2_wco AS SELECT * FROM t1 WHERE y1 < 3 WITH CHECK OPTION;
CREATE VIEW
gaussdb=# CREATE VIEW v_ins_read AS SELECT * FROM t1 WITH READ ONLY;
CREATE VIEW

-- Insert data to t1 through a view.
gaussdb=# INSERT INTO v_ins1 VALUES (1, 1);
INSERT 0 1
gaussdb=# INSERT INTO v_ins2 VALUES (5, 5);
INSERT 0 1
gaussdb=# INSERT INTO v_ins2_wco VALUES (5, 5);
ERROR:  new row violates WITH CHECK OPTION for view "v_ins2_wco"
DETAIL:  Failing row contains (5, 5).

gaussdb=# INSERT INTO v_ins_read VALUES (5, 5);
ERROR:  cannot perform a DML operation on a read-only subquery.

-- Create a multi-table view.
gaussdb=# CREATE VIEW vv_ins AS SELECT * FROM t1, t2 WHERE x1 = x2;
CREATE VIEW

-- Insert data to t1 through a view.
gaussdb=# INSERT INTO vv_ins (x1, y1) VALUES (2, 2);
INSERT 0 1

-- Delete the schema.
gaussdb=# RESET CURRENT_SCHEMA;
RESET
gaussdb=# DROP SCHEMA ins_view CASCADE;
NOTICE:  drop cascades to 7 other objects
DETAIL:  drop cascades to table ins_view.t1
drop cascades to table ins_view.t2
drop cascades to view ins_view.v_ins1
drop cascades to view ins_view.v_ins2
drop cascades to view ins_view.v_ins2_wco
drop cascades to view ins_view.v_ins_read
drop cascades to view ins_view.vv_ins
DROP SCHEMA
```

Suggestions

- VALUES

  When you run the **INSERT** statement to insert data in batches, you are advised to combine multiple records into one statement to improve data loading performance.

  Example:

  ```
  INSERT INTO sections VALUES (30, 'Administration', 31, 1900),(40, 'Development', 35, 2000), (50, 'Development' , 60 , 2001);
  ```

7.12.14 L

7.12.14.1 LOAD DATA

Description

Imports data from a file to a specified table in the database.

Precautions

- The LOAD DATA syntax is supported only in B-compatible mode (**sql_compatibility** set to **'B'**).
- The LOAD DATA syntax is the same as that of database B only when **b_format_version** is set to **'5.7'** and **b_format_dev_version** is set to **'s2'**.
- The GUC parameters related to the LOAD DATA syntax are the same as those of the COPY FROM syntax. For details, see **Precautions**.
- The LOAD DATA syntax requires the INSERT and DELETE permissions on tables.
- If the data written to a table by running **LOAD DATA** cannot be converted to the data type of the table, the import fails.
- LOAD DATA applies only to tables but not views.

Syntax

```
LOAD DATA
    [LOCAL]
    INFILE 'file_name'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [CHARACTER SET charset_name]
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
    ]
    [IGNORE number {LINES | ROWS}]
    [(col_name_or_user_var
        [, col_name_or_user_var] ...)]
    [SET col_name={expr | DEFAULT}
        [, col_name={expr | DEFAULT}] ...]
```

Parameters

- **LOCAL**

  Specifies the location of the file to be imported.

  If **LOCAL** is not specified and **file_name** is a relative path, data is imported to the data directory by default.

  If the **LOCAL** parameter is specified, **file_name** must be set to an absolute path. If **file_name** is set to a relative path, data is imported to the location of the database binary file by default, that is, *$GAUSSHOME*/**bin/**.

  NOTE

  If the imported data conflicts with the table data or the number of columns in the file is less than the number of columns in the specified table, the function of specifying **LOCAL** is the same as that of specifying **IGNORE**.

- **REPLACE | IGNORE**

  If the data to be imported conflicts with the original data in the table, specifying **REPLACE** replaces the conflicting data, and specifying **IGNORE** skips the conflicting data and continues the import. If data conflicts occur but

none of **REPLACE**, **IGNORE**, or **LOCAL** is specified, the import stops and an error is reported.

NOTE

If the number of columns in the file is less than that in the specified table, specifying **IGNORE** or **LOCAL** will assign default values to the remaining columns. If neither **IGNORE** nor **LOCAL** is specified, an error is reported.

- **PARTITION**

  If the table to be imported is a partitioned table, this parameter specifies a partition. If the data is inconsistent with the specified partition range, an error is reported.

- **CHARACTER SET**

  Specifies the encoding format of a data file. The default value is the current client encoding format.

- **FIELDS | COLUMNS**
  - **TERMINATED BY**

    Specifies the delimiter between columns. The default value is **\t**.

    NOTE

    The specified newline character cannot be the same as the delimiter.
  - **[OPTIONALLY] ENCLOSED BY**

    Specifies the quotation mark character. The default value is **''**.

    The **OPTIONALLY** parameter is optional and does not take effect.

    The quotation mark can only be a single character and cannot be a character string.
  - **ESCAPED BY**

    Specifies the escape character. The default value is **'\'**.

    The escape character can only be a single character and cannot be a character string.

- **LINES**
  - **STARTING BY**

    Specifies the style of the starting field in the data file to be imported.
  - **TERMINATED BY**

    Specifies the newline character style of the imported data file.

- **IGNORE**

  Specifies that the first *number* rows of the data file are skipped during data import.

- **col_name_or_user_var**

  Specifies an optional list of columns to be copied.

  Value range: any columns. All columns will be copied if no column list is specified.

NOTE

- The parameter for specifying columns cannot be used to specify a column repeatedly.
- When columns are specified using the LOAD DATA syntax, **col_name_or_user_var** can be specified as existing columns or user variables. If it is specified as a user variable, the GUC parameter **b_format_behavior_compat_options** must be set to **'enable_set_variables'**.

- **SET**

  Specifies the column value, which can be an expression or **DEFAULT**.

  NOTE

  - The expression does not support column names.
  - If no implicit conversion exists between the expression result type and the type of the assigned column, an error is reported.

Examples

```
-- Create a table.
gaussdb=# CREATE TABLE load_data_tbl1(load_col1 INT UNIQUE, load_col2 INT, load_col3 CHAR(10));

-- Insert a data record into a table.
gaussdb=#  INSERT INTO load_data_tbl1 VALUES(0,0,'load0');

-- Copy data from the /home/omm/load1.csv file to the load_data_tbl table, specify a column name, and set the value of the load_col3 column to load.
gaussdb=#  LOAD DATA INFILE '/home/omm/load1.csv' INTO TABLE load_data_tbl1(load_col1, load_col2) SET load_col3 = 'load';
-- The values imported to the load_col3 column later are all load.
gaussdb=#  SELECT * FROM load_data_tbl1;
 load_col1 | load_col2 | load_col3
-----------+-----------+------------
         0 |         0 | load0
         3 |         3 | load
         1 |         1 | load
         2 |         2 | load
(4 rows)

-- Copy data from the /home/omm/load2.csv file to the load_data_tbl table and specify IGNORE to ignore conflicts.
gaussdb=# LOAD DATA INFILE '/home/omm/load2.csv' IGNORE INTO TABLE load_data_tbl1;
-- Data in the load_data_tbl1 table remains unchanged, and conflicting data is skipped.
gaussdb=#  SELECT * FROM load_data_tbl1;
 load_col1 | load_col2 | load_col3
-----------+-----------+------------
         0 |         0 | load0
         3 |         3 | load
         1 |         1 | load
         2 |         2 | load
(4 rows)

-- Create a partitioned table.
gaussdb=# CREATE TABLE load_data_tbl2
(
    load_col_col1 INT,
    load_col_col2 INT
) PARTITION BY RANGE (load_col_col2)
(
    PARTITION load_p1 VALUES LESS THAN(3),
    PARTITION load_p2 VALUES LESS THAN(9),
    PARTITION load_p3 VALUES LESS THAN(MAXVALUE)
);

-- Copy data from the /home/omm/load3.csv file to the load_data_tbl2 table and specify a partition.
```



```
gaussdb=# LOAD DATA INFILE '/home/omm/load3.csv' INTO TABLE load_data_tbl2 PARTITION (load_p2);
-- Import data to the specified partition in the load_data_tbl2 table.
gaussdb=#  SELECT * FROM load_data_tbl2;
 load_col_col1 | load_col_col2
---------------+---------------
             4 |             4
             5 |             5
(2 rows)

-- Create a table.
gaussdb=# CREATE TABLE load_data_tbl3(load_col_col1 CHAR(30));

-- Copy data from the /home/omm/load4.csv file to the load_data_tbl3 table and specify FIELDS ENCLOSED BY.
gaussdb=# LOAD DATA INFILE '/home/omm/load4.csv' INTO TABLE load_data_tbl3 FIELDS ENCLOSED BY '"';
-- The double quotation marks of "load test quote" are removed, and the single quotation marks of 'load test single_quote' are retained.
gaussdb=#  select * from load_data_tbl3;
         load_col_col1
--------------------------------
 load test quote
 'load test single_quote'
(2 rows)
-- Delete the table.
gaussdb=# drop table load_data_tbl1;
gaussdb=# DROP TABLE load_data_tbl2;
gaussdb=# DROP TABLE load_data_tbl3;
```

7.12.14.2 LOAD DATA (for gs_loader)

Description

Control file syntax of gs_loader. For details about how to use gs_loader, see "Client Tools > gs_loader" in *Tool Reference*.

Syntax

```
LOAD [ DATA ]
[CHARACTERSET char_set_name]
[INFILE [directory_path] [filename ] ]
[BADFILE [directory_path] [filename ] ]
[OPTIONS(name=value)]
[{ INSERT | APPEND | REPLACE | TRUNCATE }]
INTO TABLE table_name
[{ INSERT | APPEND | REPLACE | TRUNCATE }]
[FIELDS CSV]
[TERMINATED [BY] { 'string' }]
[OPTIONALLY ENCLOSED BY { 'string' }]
[TRAILING NULLCOLS]
[ WHEN { (start:end) | column_name } {= | !=} 'string' ]
[(
col_name [ [ POSITION ({ start:end }) ]  ["sql_string"] ] | [ FILLER [column_type [external] ] ] |
[ CONSTANT "string" ] | [ SEQUENCE ( { COUNT | MAX | integer } [, incr] ) ]|[NULLIF (COL=BLANKS)]
[, ...]
)]
```

Parameters

- **CHARACTERSET**

  Specifies a character set.

  Value range: a string. Currently, the value can be **'AL32UTF8'**, **'zhs16gbk'**, or **'zhs32gb18030'**.

Note: The character set specified by **CHARACTERSET** in the control file must be the same as the encoding format of the file. Otherwise, an error is reported or garbled characters are displayed in the imported data.

- **INFILE**

  The current keyword is invalid and needs to occupy a separate line in the control file. The keyword is ignored during running. You need to specify the corresponding data file in the gs_loader command line parameters.

- **BADFILE**

  The current keyword is invalid and will be ignored during running. If no .bad file is specified in the gs_loader command, a .bad file will be generated based on the name of the corresponding control file.

- **OPTIONS**

  Only the **skip** and **rows** parameters take effect. **skip=***n* indicates that the first *n* records are skipped during import, and **rows=***n* indicates the number of rows to be imported before a commit. If both the command line and control file are specified, the command line has a higher priority.

- **INSERT | APPEND | REPLACE | TRUNCATE**

  Specifies the import mode.

  INSERT: If the table contains data, an error is reported.

  APPEND: Data is inserted directly.

  REPLACE: If the table contains data, all data is deleted and then inserted.

  TRUNCATE: If the table contains data, all data is deleted and then inserted.

  When writing a control file (.ctl), you can specify the import mode (**INSERT | APPEND | REPLACE | TRUNCATE**) before and after the INTO TABLE table_name statement. The priority is as follows: The import mode specified after the statement takes precedence over and overwrites that specified before the statement.

- **FIELDS CSV**

  Specifies that the CSV mode of COPY is used. In CSV mode, the default separator is a comma (,), and the default quotation mark is a double quotation mark (").

  CAUTION

  In the current CSV mode, quoted line feeds are considered as part of the column data.

- **table_name**

  Specifies the name (possibly schema-qualified) of an existing table.

  Value range: an existing table name

- **TERMINATED [BY] { 'string' }**

  Specifies the character string that separates columns in a file, which contains a maximum of 10 bytes.

  Value range: The value cannot include any of the following characters: \.abcdefghijklmnopqrstuvwxyz0123456789 The nul character cannot be set as a separator.

Value range: The default value is a tab character in text format or a comma in CSV format.

CAUTION

After enabling nul character compatibility (**compatible_nul** set to **true**), if the specified separator is a space character (0x20), note that the separator is the space character that exists in the data file instead of the space character converted from the nul character.

- **OPTIONALLY ENCLOSED BY { 'string' }**

  Specifies a quoted character string for a CSV file.

  The default value is double quotation marks (") only in CSV mode that is explicitly specified by the **FIELDS CSV** parameter.

  In other modes, there is no default value.

  CAUTION

  - When you set **OPTIONALLY ENCLOSED BY { 'string' }**, there should be no quotation mark on the left of the data; otherwise, the number of quotation marks on either left or right must be an odd number but they do not have to be equal.
  - Currently, **OPTIONALLY ENCLOSED BY { 'string' }** is supported only in CSV mode. If **OPTIONALLY ENCLOSED BY { 'string' }** is specified, the system enters the CSV mode by default.

- **TRAILING NULLCOLS**

  Specifies how to handle the problem that multiple columns of a row in a source data file are lost during data import.

  If one or more columns at the end of a row are null, the columns are imported to the table as null values. If this parameter is not set, an error message is displayed, indicating that the error column is empty. In this case, the data in this row is processed as an error.

- **WHEN { (start:end) | column_name } {= | !=}**

  Filters rows by character string between **start** and **end** or by column name.

  Value range: a string.

  CAUTION

  - When the GUC parameter **enable_copy_when_filler** is set to **on** (default value), data can be filtered based on the **FILLER** column. When the GUC parameter **enable_copy_when_filler** is set to **off**, this usage is not supported.
  - The WHEN condition cannot be followed by special characters such as '\0' and '\r'.

- **POSITION ({ start:end })**

  Processes columns and obtain the corresponding character strings between **start** and **end**.

- **"sql_string"**

  Processes columns and calculates column values based on column expressions. For details, see "Column Expressions" in "Client Tools > gs_loader" in *Tool Reference*.

  Value range: a string.

- **FILLER**

  Processes columns. If FILLER occurs, this column is skipped.

- **column_type [external]**

  Processes the imported data according to different data types. For details, see "Data Type" in "Client Tools > gs_loader" in *Tool Reference*.

- **CONSTANT**

  Processes columns and sets the inserted columns to constants.

  Value range: a string.

- **SEQUENCE ({ COUNT | MAX | integer } [, incr])**

  Processes columns to generate the corresponding sequence values.

  - **COUNT**: The count starts based on the number of rows in the table.
  - **MAX**: The count starts from the maximum value of this column in the table.
  - **integer**: The count starts from the specified value.
  - **incr**: indicates the increment each time.

- **NULLIF**

  Processes columns. In multi-row import scenarios, if sysdate, constant, position, or column expression is not specified after a column name, the column whose NULLIF keyword is not specified is left empty.

  Currently, only the COL POSITION() CHAR NULLIF (COL=BLANKS) syntax is supported. For details, see "NULLIF Usage Cases" in "Client Tools > gs_loader" in *Tool Reference*.

7.12.14.3 LOCK

Description

LOCK TABLE obtains a table-level lock.

GaussDB always tries to select the lock mode with minimum constraints when automatically requesting a lock for a statement referenced by a table. Use **LOCK** if users need a more strict lock mode. For example, suppose an application runs a transaction at the Read Committed isolation level and needs to ensure that data in a table remains stable in the duration of the transaction. To achieve this, you could obtain **SHARE** lock mode over the table before the query. This will prevent concurrent data changes and ensure subsequent reads of the table see a stable view of committed data. It is because the **SHARE** lock mode conflicts with the **ROW EXCLUSIVE** lock acquired by writers, and your **LOCK TABLE name IN SHARE MODE** statement will wait until any concurrent holders of **ROW**

EXCLUSIVE mode locks commit or roll back. Therefore, once you obtain the lock, there are no uncommitted writes outstanding; furthermore none can begin until you release the lock.

Precautions

- **LOCK TABLE** is useless outside a transaction block: the lock would remain held only to the completion of the statement. If **LOCK TABLE** is out of any transaction block, an error is reported.
- If no lock mode is specified, then **ACCESS EXCLUSIVE**, the most restrictive mode, is used.
- LOCK TABLE ... **IN ACCESS SHARE MODE** requires the **SELECT** permission on the target table. All other forms of **LOCK** require table-level **UPDATE** and/or the **DELETE** permission.
- There is no **UNLOCK TABLE** statement. Locks are always released at transaction end.
- **LOCK TABLE** only deals with table-level locks, and so the mode names involving **ROW** are all misnomers. These mode names should generally be read as indicating the intention of the user to acquire row-level locks within the locked table. Also, **ROW EXCLUSIVE** mode is a shareable table lock. Note that all lock modes have the same semantics as long as LOCK TABLE is involved. The only difference lies in whether locks conflict with each other. For details about the rules, see **Table 7-224**.
- If the xc_maintenance_mode parameter is not enabled, an error is reported when an ACCESS EXCLUSIVE lock is applied for a system catalog.

Syntax

```
LOCK [ TABLE ] {[ ONLY ] name [, ...]| {name [ * ]} [, ...]}
    [ IN {ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE | SHARE | SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE} MODE ]
    [ NOWAIT ];
```

Parameters

Table 7-224 Lock mode conflicts

Requested Lock Mode/ Current Lock Mode	ACCESS SHARE	ROW SHARE	ROW EXCLUSIVE	SHARE UPDATE EXCLUSIVE	SHARE	SHARE ROW EXCLUSIVE	EXCLUSIVE	ACCESS EXCLUSIVE
ACCESS SHARE	-	-	-	-	-	-	-	X
ROW SHARE	-	-	-	-	-	-	X	X

Requested Lock Mode/ Current Lock Mode	ACCESS SHARE	ROW SHARE	ROW EXCLUSIVE	SHARE UPDATE EXCLUSIVE	SHARE	SHARE ROW EXCLUSIVE	EXCLUSIVE	ACCESS EXCLUSIVE
ROW EXCLUSIVE	-	-	-	-	X	X	X	X
SHARE UPDATE EXCLUSIVE	-	-	-	X	X	X	X	X
SHARE	-	-	X	X	-	X	X	X
SHARE ROW EXCLUSIVE	-	-	X	X	X	X	X	X
EXCLUSIVE	-	X	X	X	X	X	X	X
ACCESS EXCLUSIVE	X	X	X	X	X	X	X	X

LOCK parameters are as follows:

- **name**

  Specifies the name (optionally schema-qualified) of an existing table to lock.

  Tables are locked one-by-one in the order specified in the **LOCK TABLE** statement.

  Value range: an existing table name

  NOTE

  You can use database links to perform operations on remote tables. For details, see **DATABASE LINK**.

- **ONLY**

  If **ONLY** is specified, only that table is locked. If it is not specified, the table and all its sub-tables are locked.

- **ACCESS SHARE**

  Conflicts with the ACCESS EXCLUSIVE lock mode only.

The **SELECT** statement acquires a lock of this mode on referenced tables. Typically, any command that reads a table without modifying it acquires this lock mode.

- **ROW SHARE**

  It conflicts with the EXCLUSIVE and ACCESS EXCLUSIVE lock modes.

  SELECT FOR UPDATE and **SELECT FOR SHARE** automatically acquire the **ROW SHARE** lock on the target table and add the **ACCESS SHARE** lock to other referenced tables except **FOR SHARE** and **FOR UPDATE**.

- **ROW EXCLUSIVE**

  Allows concurrent read of a table but does not allow modification of data in the table. **ROW EXCLUSIVE** is the same as **ROW SHARE**. **UPDATE**, **DELETE**, and **INSERT** automatically acquire this lock on the target table and add the **ACCESS SHARE** lock to other referenced tables. Generally, all statements that modify table data acquire the **ROW EXCLUSIVE** lock for tables.

- **SHARE UPDATE EXCLUSIVE**

  Protects a table against concurrent schema changes and VACUUM runs.

  The VACUUM (without FULL), ANALYZE, and CREATE INDEX CONCURRENTLY statements automatically request this lock.

- **SHARE**

  Allows concurrent queries of a table but does not allow modification of the table.

  Acquired by **CREATE INDEX** (without **CONCURRENTLY**).

- **SHARE ROW EXCLUSIVE**

  Protects a table against concurrent data changes, and is self-exclusive so that only one session can hold it at a time.

  No SQL statements automatically acquire this lock mode.

- **EXCLUSIVE**

  Allows concurrent queries of the target table but does not allow any other operations.

  This mode allows only concurrent **ACCESS SHARE** locks; that is, only reads from the table can proceed in parallel with a transaction holding this lock mode.

  No SQL statements automatically acquire this lock mode on user tables. However, it will be acquired on some system catalogs in case of some operations.

- **ACCESS EXCLUSIVE**

  Guarantees that the holder is the only transaction accessing the table in any way.

  The ALTER TABLE, DROP TABLE, TRUNCATE and REINDEX statements automatically request this lock.

  This is also the default lock mode for **LOCK TABLE** statements that do not specify a mode explicitly.

- **NOWAIT**

  Specifies that **LOCK TABLE** should not wait for any conflicting locks to be released: if the specified lock(s) cannot be acquired immediately without waiting, the transaction is aborted.

If a table-level lock is obtained without specifying **NOWAIT** and other mutex locks exist, the system waits for other locks to be released.

Examples

- SHARE ROW EXCLUSIVE

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create the tpcds.reason table.
gaussdb=# CREATE TABLE tpcds.reason (
r_reason_sk         INTEGER      NOT NULL,
r_reason_id         CHAR(16)     NOT NULL,
r_reason_desc       INTEGER
);

-- Insert multiple records into the table.
gaussdb=# INSERT INTO tpcds.reason VALUES (1, 'AAAAAAAABAAAAAAA', '18'),(5,
'AAAAAAAACAAAAAAA', '362'),(7, 'AAAAAAAADAAAAAAA', '585');

-- Create a table named reason_t1.
gaussdb=# CREATE TABLE tpcds.reason_t1 AS TABLE tpcds.reason;

-- Start a transaction.
gaussdb=# START TRANSACTION;

-- Use the SHARE ROW EXCLUSIVE MODE lock mode to lock a table.
gaussdb=# LOCK TABLE tpcds.reason_t1 IN SHARE ROW EXCLUSIVE MODE;

-- If you perform the DELETE operation on another device, the operation is blocked.
gaussdb=# DELETE FROM tpcds.reason_t1 WHERE r_reason_sk = 7;

-- End the transaction and release the lock.
gaussdb=# COMMIT;
```

- ROW EXCLUSIVE

```
-- Start a transaction.
gaussdb=# START TRANSACTION;

-- After the UPDATE statement is executed, request a ROW EXCLUSIVE lock on the referenced table.
gaussdb=# UPDATE tpcds.reason_t1 SET r_reason_desc=180 WHERE r_reason_sk=1;

-- When the ALTER TABLE statement is executed on another device, the operation is blocked.
gaussdb=# ALTER TABLE tpcds.reason_t1 ADD r_reason_asc int;

-- End the transaction and release the lock.
gaussdb=# COMMIT;
```

- ACCESS EXCLUSIVE

```
-- Start a transaction.
gaussdb=# START TRANSACTION;

-- After the TRUNCATE statement is executed, request an ACCESS SHARE lock on the referenced table.
gaussdb=# TRUNCATE tpcds.reason_t1;

-- When the SELECT statement is executed on another device, the operation is blocked.
gaussdb=# SELECT * FROM tpcds.reason_t1;

-- End the transaction and release the lock.
gaussdb=# COMMIT;

-- Delete the tpcds.reason_t1 table.
gaussdb=# DROP TABLE tpcds.reason_t1;

-- Delete the tpcds.reason table.
gaussdb=# DROP TABLE tpcds.reason;

-- Delete a schema.
gaussdb=# DROP SCHEMA tpcds CASCADE;
```


7.12.14.4 LOCK BUCKETS

This function is not supported in the current version.

7.12.15 M

7.12.15.1 MARK BUCKETS

This function is not supported in the current version.

7.12.15.2 MERGE INTO

Function

MERGE INTO conditionally matches data in a target table with that in a source table. If data matches, UPDATE is executed on the target table; if data does not match, INSERT is executed. You can use this syntax to run UPDATE and INSERT at a time for convenience.

Precautions

You have the INSERT and UPDATE permissions for the target table and the SELECT permission for the source table.

Syntax

```
MERGE [/*+ plan_hint */] INTO table_name [ partition_clause ] [ [ AS ] alias ]
USING { { table_name | view_name } | subquery } [ [ AS ] alias ]
ON ( condition )
[
  WHEN MATCHED THEN
  UPDATE SET { column_name = { expression | subquery | DEFAULT } |
          ( column_name [, ...] ) = ( { expression | subquery | DEFAULT } [, ...] ) } [, ...]
  [ WHERE condition ]
]
[
  WHEN NOT MATCHED THEN
  INSERT { DEFAULT VALUES |
  [ ( column_name [, ...] ) ] VALUES ( { expression | subquery | DEFAULT } [, ...] ) [, ...] [ WHERE condition ] }
];
NOTICE: 'subquery' in the UPDATE and INSERT clauses are only available in CENTRALIZED mode!
```

Parameter Description

- plan_hint clause

  Follows the MERGE keyword in the **/*+ */** format. It is used to optimize the plan of a MERGE statement block. For details, see **Hint-based Tuning**. In each statement, only the first **/*+** *plan_hint* ***/** comment block takes effect as a hint. Multiple hints can be written.

- INTO clause

  Specifies the target table that is being updated or has data being inserted.

- **table_name**

  Specifies the name of the target table.

- **partition_clause**

Performs MERGE operations on a specified partition.

```
PARTITION { ( partition_name ) | FOR ( partition_value [, ...] ) } |
SUBPARTITION { ( subpartition_name ) | FOR ( subpartition_value [, ...] ) }
```

For details about the keywords, see **SELECT**.

If the value of the VALUE clause is inconsistent with that of the specified partition, an exception is displayed.

For details, see **CREATE TABLE SUBPARTITION**.

- **alias**

  Specifies the alias of the target table.

  Value range: a string. It must comply with the **naming convention**.

- USING clause

  Specifies the source table, which can be a table, view, or subquery.

- ON clause

  Specifies the condition used to match data between the source and target tables. Columns in the condition cannot be updated.

- WHEN MATCHED clause

  Performs UPDATE if data in the source table matches that in the target table based on the condition.

  System catalogs and system columns cannot be updated.

- WHEN NOT MATCHED clause

  Performs INSERT if data in the source table does not match that in the target table based on the condition.

  An INSERT clause cannot contain multiple VALUES.

  The order of WHEN MATCHED and WHEN NOT MATCHED clauses can be reversed. One of them can be used by default, but they cannot be both used at one time. Two WHEN MATCHED or WHEN NOT MATCHED clauses cannot be specified at the same time.

- **DEFAULT**

  Specifies the default value of a column.

  The value is **NULL** if no default value is assigned to it.

- **WHERE condition**

  Specifies the conditions for the UPDATE and INSERT clauses. The two clauses will be executed only when the conditions are met. The default value can be used. System columns cannot be referenced in WHERE condition. You are advised not to use numeric types such as int as conditions, because such types can be implicitly converted to bool values (non-zero values are implicitly converted to **true** and **0** is implicitly converted to **false**), which may cause unexpected results.

Examples

```
-- Create the target table products and source table newproducts, and insert data to them.
gaussdb=# CREATE TABLE products
(
product_id INTEGER,
product_name VARCHAR2(60),
category VARCHAR2(60)
);
```



```
gaussdb=# INSERT INTO products VALUES (1501, 'vivitar 35mm', 'electrncs');
gaussdb=# INSERT INTO products VALUES (1502, 'olympus is50', 'electrncs');
gaussdb=# INSERT INTO products VALUES (1600, 'play gym', 'toys');
gaussdb=# INSERT INTO products VALUES (1601, 'lamaze', 'toys');
gaussdb=# INSERT INTO products VALUES (1666, 'harry potter', 'dvd');

gaussdb=# CREATE TABLE newproducts
(
product_id INTEGER,
product_name VARCHAR2(60),
category VARCHAR2(60)
);

gaussdb=# INSERT INTO newproducts VALUES (1502, 'olympus camera', 'electrncs');
gaussdb=# INSERT INTO newproducts VALUES (1601, 'lamaze', 'toys');
gaussdb=# INSERT INTO newproducts VALUES (1666, 'harry potter', 'toys');
gaussdb=# INSERT INTO newproducts VALUES (1700, 'wait interface', 'books');

-- Run MERGE INTO.
gaussdb=# MERGE INTO products p
USING newproducts np
ON (p.product_id = np.product_id)
WHEN MATCHED THEN
  UPDATE SET p.product_name = np.product_name, p.category = np.category WHERE p.product_name !=
'play gym'
WHEN NOT MATCHED THEN
  INSERT VALUES (np.product_id, np.product_name, np.category) WHERE np.category = 'books';
MERGE 4

-- Query updates.
gaussdb=# SELECT * FROM products ORDER BY product_id;
 product_id |  product_name  | category
------------+----------------+-----------
       1501 | vivitar 35mm   | electrncs
       1502 | olympus camera | electrncs
       1600 | play gym       | toys
       1601 | lamaze         | toys
       1666 | harry potter   | toys
       1700 | wait interface | books
(6 rows)

-- Delete the table.
gaussdb=# DROP TABLE products;
gaussdb=# DROP TABLE newproducts;
```

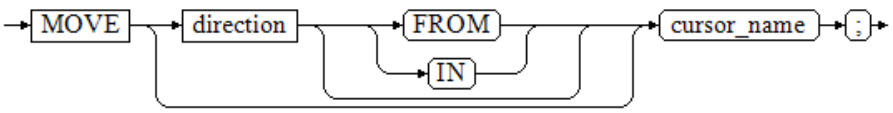
7.12.15.3 MOVE

Description

MOVE repositions a cursor without retrieving any data. MOVE works exactly like the **FETCH** statement, except MOVE only repositions the cursor and does not return rows.

Syntax

```
MOVE [ direction [ FROM | IN ] ] cursor_name;
```

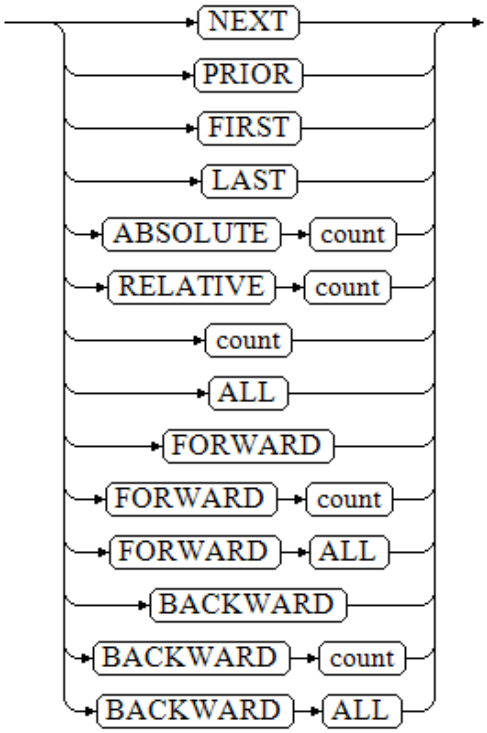


The direction clause specifies optional parameters.

```
NEXT
  | PRIOR
```



```
| FIRST
| LAST
| ABSOLUTE count
| RELATIVE count
| count
| ALL
| FORWARD
| FORWARD count
| FORWARD ALL
| BACKWARD
| BACKWARD count
| BACKWARD ALL
```



Parameters

The parameters of MOVE and FETCH are the same. For details, see **Parameters** in section "FETCH."

NOTE

On successful completion, a MOVE statement returns a tag of the form **MOVE count**. The **count** is the number of rows that a FETCH statement with the same parameters would have returned (possibly zero).

Examples

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE tbl_test(c1 int);
gaussdb=# INSERT INTO tbl_test VALUES (generate_series(1,20));

-- Set up cursor1.
gaussdb=# BEGIN;
gaussdb=# CURSOR cursor1 FOR SELECT * FROM tbl_test ORDER BY 1;

-- Run the MOVE command to move the cursor backwards by five rows. No result is returned.
gaussdb=# MOVE FORWARD 5 FROM cursor1;
```



```
MOVE 5

-- Run the FETCH command to retrieve two rows of data.
gaussdb=# FETCH FORWARD 2 FROM cursor1;
 c1
----
  6
  7
(2 rows)

-- Close the cursor and end the transaction.
gaussdb=# CLOSE cursor1;
gaussdb=# END;

-- Delete.
gaussdb=# DROP TABLE tbl_test;
```

Helpful Links

CLOSE and **FETCH**

7.12.16 P

7.12.16.1 PREDICT BY

Description

Uses a trained model to perform prediction tasks.

Precautions

The name of the invoked model can be viewed in the gs_model_warehouse system catalog.

Syntax

```
PREDICT BY model_name (FEATURES attribute [, attribute]...);
```

Parameters

- **model_name**

  Name of the model of a speculative task.

  Value range: a string. It must comply with the **naming convention**.

- **attribute**

  Name of the input feature column of a speculative task.

  Value range: a string. It must comply with the **naming convention**.

Examples

```
-- Create a data table.
gaussdb=# CREATE TABLE houses (
id INTEGER,
tax INTEGER,
bedroom INTEGER,
bath DOUBLE PRECISION,
price INTEGER,
size INTEGER,
```



```
lot INTEGER,
mark text
);

-- Insert training data.
gaussdb=# INSERT INTO houses(id, tax, bedroom, bath, price, size, lot, mark) VALUES
(1,590,2,1,50000,770,22100,'a+'),
(2,1050,3,2,85000,1410,12000,'a+'),
(3,20,2,1,22500,1060,3500,'a-'),
(4,870,2,2,90000,1300,17500,'a+'),
(5,1320,3,2,133000,1500,30000,'a+'),
(6,1350,2,1,90500,850,25700,'a-'),
(7,2790,3,2.5,260000,2130,25000,'a+'),
(8,680,2,1,142500,1170,22000,'a-'),
(9,1840,3,2,160000,1500,19000,'a+'),
(10,3680,4,2,240000,2790,20000,'a-'),
(11,1660,3,1,87000,1030,17500,'a+'),
(12,1620,3,2,118500,1250,20000,'a-'),
(13,3100,3,2,140000,1760,38000,'a+'),
(14,2090,2,3,148000,1550,14000,'a-'),
(15,650,3,1.5,65000,1450,12000,'a-');

-- Train the model.
gaussdb=# CREATE MODEL price_model USING logistic_regression
 FEATURES size, lot
 TARGET mark
 FROM HOUSES
 WITH learning_rate=0.88, max_iterations=default;

-- Predict.
gaussdb=# SELECT id, PREDICT BY price_model (FEATURES size,lot) FROM houses;

-- Delete the model.
gaussdb=# DROP MODEL price_model;

-- Delete the table.
gaussdb=# DROP TABLE houses;
```

Helpful Links

CREATE MODEL and **DROP MODEL**

7.12.16.2 PREPARE

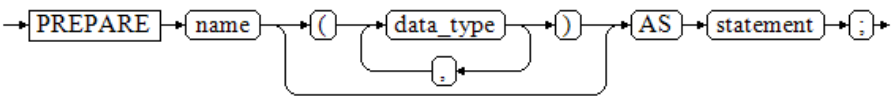
Description

Creates a prepared statement.

A prepared statement is a performance optimizing object on the server. When the **PREPARE** statement is executed, the specified query is parsed, analyzed, and rewritten. When **EXECUTE** is executed, the prepared statement is planned and executed. This avoids repetitive parsing and analysis. After the **PREPARE** statement is created, it exists throughout the database session. Once it is created (even if in a transaction block), it will not be deleted when a transaction is rolled back. It can only be deleted by explicitly invoking **DEALLOCATE** or automatically deleted when the session ends.

Syntax

```
PREPARE name [ ( data_type [, ...] ) ] AS statement;
```



Parameters

- **name**

  Specifies the name of a prepared statement. It must be unique in the session.
- **data_type**

  Specifies the type of an argument.
- **statement**

  Specifies a SELECT, INSERT, UPDATE, DELETE, MERGE INTO, or VALUES statement.

Examples

See **Examples** in section "EXECUTE."

Helpful Links

DEALLOCATE and **7.13.10.1-EXECUTE**

7.12.16.3 PREPARE TRANSACTION

Description

Prepares the current transaction for two-phase commit.

After this statement, the transaction is no longer associated with the current session; instead, its state is fully stored on disk, and there is a high probability that it can be committed successfully, even if a database crash occurs before the commit is requested.

Once prepared, a transaction can later be committed or rolled back with **COMMIT PREPARED** or **ROLLBACK PREPARED**, respectively. Those statements can be issued from any session, not only the one that executed the original transaction.

From the point of view of the issuing session, **PREPARE TRANSACTION** is not unlike a **ROLLBACK** statement: after executing it, there is no active current transaction, and the effects of the prepared transaction are no longer visible. (The effects will become visible again if the transaction is committed.)

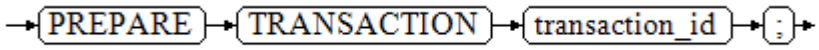
If the **PREPARE TRANSACTION** statement fails for any reason, it becomes a **ROLLBACK** and the current transaction is canceled.

Precautions

- The transaction function is maintained automatically by the database, and should be not visible to users.
- When running the **PREPARE TRANSACTION** statement, increase the value of **max_prepared_transactions** in configuration file **gaussdb.conf**. You are advised to set it to a value not less than that of **max_connections** so that one pending prepared transaction is available for each session.

Syntax

```
PREPARE TRANSACTION transaction_id;
```

Parameters

- **transaction_id**

  Specifies an arbitrary identifier that later identifies this transaction for **COMMIT PREPARED** or **ROLLBACK PREPARED**. The identifier must be different from those for current prepared transactions.

  Value range: The identifier must be written as a string literal, and must be less than 200 bytes long.

Examples

```
-- Start.
gaussdb=# BEGIN;
BEGIN

-- Prepare a transaction whose identifier is trans_test.
gaussdb=# PREPARE TRANSACTION 'trans_test';
PREPARE TRANSACTION

-- Cancel the transaction whose identifier is trans_test.
gaussdb=# ROLLBACK PREPARED 'trans_test';
ROLLBACK PREPARED
```

Helpful Links

COMMIT PREPARED and **ROLLBACK PREPARED**

7.12.16.4 PURGE

Description

The **PURGE** statement can be used to:

- Clear tables or indexes from the recycle bin and release all space related to the objects.
- Clear the recycle bin.
- Clear the objects of a specified tablespace in the recycle bin.

Precautions

- The PURGE operation supports tables (**PURGE TABLE**), indexes (**PURGE INDEX**), and recycle bins (**PURGE RECYCLEBIN**).
- The permission requirements for performing the PURGE operation are as follows:
  - **PURGE TABLE**: The user must be the owner of the table and must have the USAGE permission on the schema to which the table belongs. When the separation of duties is disabled, system administrators have this permission by default.
  - **PURGE INDEX**: The user must be the owner of the index and have the USAGE permission on the schema to which the index belongs. When the

separation of duties is disabled, system administrators have this permission by default.

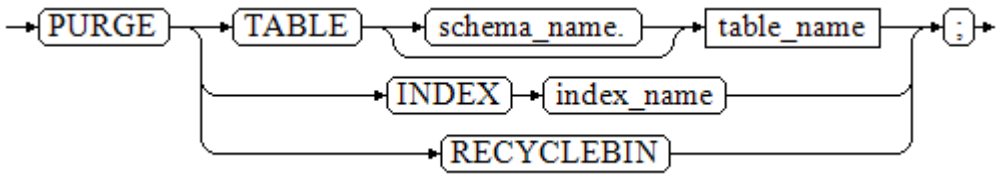
- **PURGE RECYCLEBIN**: Common users can clear only the objects owned by themselves in the recycle bin. In addition, the user must have the USAGE permission of the schema to which the objects belong. When the separation of duties is disabled, system administrators can clear all objects in the recycle bin by default.

Prerequisites

- The **enable_recyclebin** parameter has been enabled to enable the recycle bin. Contact the administrator for details about how to use the parameter.
- The **recyclebin_retention_time** parameter has been set for specifying the retention period of objects in the recycle bin. The objects will be automatically deleted after the retention period expires. Contact the administrator for details about how to use the parameter.

Syntax

```
PURGE { TABLE [schema_name.]table_name
      | INDEX index_name
      | RECYCLEBIN
    };
```



Parameters

- **schema_name**

  Specifies the schema name.

- **TABLE [schema_name.] table_name**

  Clears a specified table in the recycle bin. The table can be schema-qualified.

- **INDEX index_name**

  Clears a specified index in the recycle bin.

- **RECYCLEBIN**

  Clears all objects in the recycle bin.

Examples

```
-- Create the reason_t1 table.
gaussdb=# CREATE TABLE reason_t1(
    r_reason_sk    integer,
    r_reason_id    character(16),
    r_reason_desc  character(100)
) WITH(STORAGE_TYPE = ustore);

-- Create the reason_t2 table.
gaussdb=# CREATE TABLE reason_t2(
    r_reason_sk    integer,
    r_reason_id    character(16),
```



```
   r_reason_desc  character(100)
) WITH(STORAGE_TYPE = ustore);

-- Add indexes to the reason_t1 and reason_t2 tables.
gaussdb=#  CREATE INDEX idx_t1 on reason_t1(r_reason_id);
gaussdb=#  CREATE INDEX idx_t2 on reason_t2(r_reason_id);
gaussdb=#  DROP TABLE reason_t1;
gaussdb=#  DROP TABLE reason_t2;

-- View the recycle bin.
gaussdb=#  SELECT rcyname,rcyoriginname,rcytablespace FROM GS_RECYCLEBIN;
          rcyname           | rcyoriginname | rcytablespace
----------------------------+---------------+---------------
 BIN$31C94EB4207$8001$0==$0 | reason_t1     |             0
 BIN$31C94EB420D$8001$0==$0 | idx_t1        |             0
 BIN$31C94EB420A$8004$0==$0 | reason_t2     |             0
 BIN$31C94EB420E$8004$0==$0 | idx_t2        |             0
(4 rows)

-- Purge the table.
gaussdb=#  PURGE TABLE reason_t1;
gaussdb=#  SELECT rcyname,rcyoriginname,rcytablespace FROM GS_RECYCLEBIN;
          rcyname           | rcyoriginname | rcytablespace
----------------------------+---------------+---------------
 BIN$31C94EB420A$8004$0==$0 | reason_t2     |             0
 BIN$31C94EB420E$8004$0==$0 | idx_t2        |             0
(2 rows)

-- Purge the index.
gaussdb=# PURGE INDEX idx_t2;
gaussdb=# SELECT rcyname,rcyoriginname,rcytablespace FROM GS_RECYCLEBIN;
          rcyname           | rcyoriginname | rcytablespace
----------------------------+---------------+---------------
 BIN$31C94EB420A$8004$0==$0 | reason_t2     |             0
(1 row)

-- Purge all objects in the recycle bin.
gaussdb=#  PURGE recyclebin;
gaussdb=#  SELECT rcyname,rcyoriginname,rcytablespace FROM GS_RECYCLEBIN;
        rcyname         | rcyoriginname | rcytablespace
------------------------+---------------+---------------
(0 rows)
```

7.12.17 R

7.12.17.1 REASSIGN OWNED

Description

Changes the owner of the database object.

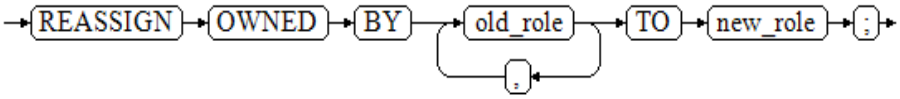
REASSIGN OWNED changes the database object owner of an old role to a new role.

Precautions

- REASSIGN OWNED is often executed before role deletion.
- To run the REASSIGN OWNED statement, you must have the permissions of the original and target roles.

Syntax

```
REASSIGN OWNED BY old_role [, ...] TO new_role;
```

Parameters

- **old_role**

  Specifies the role name of the old owner.

- **new_role**

  Specifies the role name of the new owner. Note: Only the initial user can use the REASSIGN OWNED syntax to change the owner to the initial user.

Examples

```
-- Create the test_jim and test_tom users.
gaussdb=# CREATE USER test_jim PASSWORD '********';
gaussdb=# CREATE USER test_tom PASSWORD '********';

-- View the user with the same name as owner of the automatically created schema.
gaussdb=# \dn test*
   List of schemas
   Name   |  Owner
----------+----------
 test_jim | test_jim
 test_tom | test_tom
(2 rows)

-- Change the owner of all database objects owned by test_jim to test_tom.
gaussdb=# REASSIGN OWNED BY test_jim TO test_tom;

-- View the schema information. The owner of the test_jim schema is changed to test_tom.
gaussdb=# \dn test*
   List of schemas
   Name   |  Owner
----------+----------
 test_jim | test_tom
 test_tom | test_tom
(2 rows)

-- Delete the test_jim and test_tom users.
gaussdb=# DROP USER test_jim, test_tom CASCADE;
```

Helpful Links

DROP OWNED

7.12.17.2 REFRESH INCREMENTAL MATERIALIZED VIEW

Description

REFRESH INCREMENTAL MATERIALIZED VIEW refreshes a materialized view in incremental mode.

Precautions

- Incremental refresh supports only fast-refresh materialized views.
- To refresh a materialized view, you must have the SELECT permission on the base table.

Syntax

```
REFRESH INCREMENTAL MATERIALIZED VIEW mv_name;
```

Parameters

- **mv_name**

  Name of the materialized view to be refreshed.

Examples

```
-- Create an ordinary table.
gaussdb=# CREATE TABLE my_table (c1 int, c2 int)
WITH(STORAGE_TYPE=ASTORE);

-- Create a fast-refresh materialized view.
gaussdb=# CREATE INCREMENTAL MATERIALIZED VIEW my_imv AS SELECT * FROM my_table;

-- Write data to the base table.
gaussdb=# INSERT INTO my_table VALUES(1,1),(2,2);

-- Fast refresh the fast-refresh materialized view my_imv.
gaussdb=# REFRESH INCREMENTAL MATERIALIZED VIEW my_imv;

-- View the fast-refresh materialized view.
gaussdb=# SELECT * FROM my_imv;
 c1 | c2
----+----
  1 |  1
  2 |  2
(2 rows)

-- Delete the fast-refresh materialized view.
gaussdb=# DROP MATERIALIZED VIEW my_imv;

-- Delete the my_table table.
gaussdb=# DROP TABLE my_table;
```

Helpful Links

ALTER MATERIALIZED VIEW, **CREATE INCREMENTAL MATERIALIZED VIEW**, **CREATE MATERIALIZED VIEW**, **CREATE TABLE**, **DROP MATERIALIZED VIEW**, and **REFRESH MATERIALIZED VIEW**

7.12.17.3 REFRESH MATERIALIZED VIEW

Function

REFRESH MATERIALIZED VIEW refreshes materialized views in complete-refresh mode.

Precautions

- Full refreshing can be performed on both complete- and fast-refresh materialized views.
- To refresh a materialized view, you must have the SELECT permission on the base table.

Syntax

```
REFRESH MATERIALIZED VIEW mv_name;
```


Parameter Description

- **mv_name**

  Name of the materialized view to be refreshed.

Examples

```
-- Create an ordinary table.
gaussdb=# CREATE TABLE my_table (c1 int, c2 int) WITH(STORAGE_TYPE=ASTORE);

-- Create a complete-refresh materialized view.
gaussdb=# CREATE MATERIALIZED VIEW my_mv AS SELECT * FROM my_table;

-- Create a fast-refresh materialized view.
gaussdb=# CREATE INCREMENTAL MATERIALIZED VIEW my_imv AS SELECT * FROM my_table;

-- Write data to the base table.
gaussdb=# INSERT INTO my_table VALUES(1,1),(2,2);

-- Completely refresh the complete-refresh materialized view my_mv.
gaussdb=# REFRESH MATERIALIZED VIEW my_mv;

-- Completely refresh the fast-refresh materialized view my_imv.
gaussdb=# REFRESH MATERIALIZED VIEW my_imv;

-- Delete a fast-refresh materialized view.
gaussdb=# DROP MATERIALIZED VIEW my_imv;

-- Delete a complete-refresh materialized view.
gaussdb=# DROP MATERIALIZED VIEW my_mv;

-- Delete the my_table table.
gaussdb=# DROP TABLE my_table;
```

Helpful Links

ALTER MATERIALIZED VIEW, **CREATE INCREMENTAL MATERIALIZED VIEW**, **CREATE MATERIALIZED VIEW**, **CREATE TABLE**, **DROP MATERIALIZED VIEW**, and **REFRESH INCREMENTAL MATERIALIZED VIEW**

7.12.17.4 REINDEX

Description

Rebuilds an index using the data stored in the index's table, replacing the old copy of the index.

There are several scenarios in which **REINDEX** can be used:

- An index has become corrupted, and no longer contains valid data.
- An index has become "bloated", that is, it contains many empty or nearly-empty pages.
- You have altered a storage parameter (such as a fill factor) for an index, and wish that the change takes full effect.
- An index build with the **CONCURRENTLY** option failed, leaving an "invalid" index.

Precautions

- **REINDEX DATABASE** and **REINDEX SYSTEM** type cannot be performed in transaction blocks. Currently, REINDEX operations cannot be performed on materialized views.
- If the index contains the **lpi_parallel_method** option and the value is **PARTITION**, and the **parallel_workers** value of the index's table is greater than 0, the index cannot be rebuilt in parallel. If the index does not contain the **lpi_parallel_method** option or the value of the option is set to **AUTO**, page-level parallel index is rebuilt by default. For details, see **LPI_PARALLEL_METHOD**.

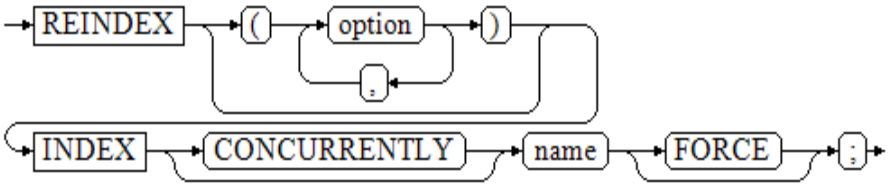
Syntax

- Rebuild a general index.

```
REINDEX { INDEX | TABLE | DATABASE | SYSTEM } [CONCURRENTLY] name [ FORCE ];
```

- Rebuild the index and convert the type.

```
REINDEX [ ( option [, ...] ) ] { INDEX } [ CONCURRENTLY ] name [ FORCE ];
```



- Rebuild an index partition.

```
REINDEX { INDEX | TABLE} name
    PARTITION partition_name [ FORCE ];
```

Parameters

- **INDEX**

  Rebuilds the specified index.

- **TABLE**

  Rebuilds all indexes of a specified table. If a table has a TOAST table, the table will also be reindexed. If an index on the table has been invalidated by running **alter unusable**, the index cannot be rebuilt. Indexes in the TOAST table cannot be rebuilt when specifying the CONCURRENTLY option.

- **DATABASE**

  Rebuilds all indexes within the current database. Indexes in the TOAST table within the current database cannot be rebuilt when specifying the CONCURRENTLY option.

- **SYSTEM**

  Rebuilds all indexes on system catalogs within the current database. Indexes on user tables are not processed.

- **option**

  Currently, only **CROSSBUCKET** is supported, and the value can only be **ON** or **OFF**. This parameter is used to determine whether the index of a hash bucket table is converted to a cross-bucket index (CBI) or local-bucket index (LBI). This conversion supports only the indexes on a distributed hash bucket table and does not support GSIs.

- **CONCURRENTLY**

  Rebuilds an index (with ShareUpdateExclusiveLock) in non-blocking DML mode. When an index is rebuilt, other statements cannot access the table on which the index depends. If this keyword is specified, DML is not blocked during the recreation. Indexes in system catalogs cannot be rebuilt online. REINDEX INTERNAL TABLE CONCURRENTLY, REINDEX SYSTEM CONCURRENTLY, and REINDEX INVALID INDEX CONCURRENTLY are not supported. When REINDEX DATABASE CONCURRENTLY is executed, all indexes on user tables in the current database are rebuilt online (indexes on system catalogs are not processed). REINDEX CONCURRENTLY cannot be executed within a transaction. Online index rebuilding supports only B-tree and UB-tree indexes, common indexes, global indexes, and local indexes. PCR UB-tree indexes, level-2 partitions, and GSIs are not supported. Online concurrent index rebuilding supports only common indexes, global indexes, and local indexes of Astore and Ustore. Other online index rebuilding specifications are inherited from the current version. If online index rebuilding fails, the system automatically clears new indexes to prevent resource occupation in scenarios such as manual cancellation, duplicate unique index key values, insufficient resources, thread startup failure, and lock timeout. If the system cannot automatically clear invalid new indexes (for example, the database breaks down, PATAL, or PANIC), you need to manually clear invalid new indexes (using the DROP INDEX statement) and temporary tables (using the DROP TABLE statement) as soon as possible to prevent more resources from being occupied. Generally, the extension of an invalid index name is **_ccnew**. The execution of REINDEX INDEX CONCURRENTLY adds a four-level session lock to the table and its first several phases are similar to those of CREATE INDEX CONCURRENTLY. Therefore, the execution may be suspended or deadlocked, which is similar to that of CREATE INDEX CONCURRENTLY. For example, if two sessions perform the REINDEX CONCURRENTLY operation on the same index or table at the same time, a deadlock occurs. For details, see **CONCURRENTLY**.

  NOTE

  This keyword is specified when an index is rebuilt. For Astore, you need to complete two full table scans for building. During the first scan, a new index is created without blocking read and write operations. During the second scan, the changes in the first scan are merged and updated. For Ustore, you need to complete a full table scan. During the scan, data generated by concurrent DML operations is inserted into the temporary table named **index_oid_cctmp**. After the scan is complete, you merge the temporary table to the new index suffixed with **_ccnew{n}**, delete the temporary table, exchange the old and new indexes, mark the old index as dead, enable the new index, and rebuild the index.

- **name**

  Specifies the name of the index, table, or database whose index needs to be rebuilt. Tables and indexes can be schema-qualified.

  NOTE

  REINDEX DATABASE and **SYSTEM** can create indexes for only the current database. Therefore, **name** must be the same as the current database name.

- **FORCE**

  Discarded parameter. It is currently reserved for compatibility with earlier versions.

- **partition_name**

  Specifies the name of the partition or index partition to be rebuilt.

  Value range:

  - If **REINDEX INDEX** is used, specify the name of an index partition.
  - If it is **REINDEX TABLE**, specify the name of a partition.

NOTICE

REINDEX DATABASE and **REINDEX SYSTEM** type cannot be performed in transaction blocks.

Examples

```
-- Create the table tbl_test and insert data into the tables.
gaussdb=# CREATE TABLE tbl_test(c1 int,c2 varchar);
gaussdb=# INSERT INTO tbl_test VALUES (1, 'AAAAAAA'),(5, 'AAAAAAB'),(10, 'AAAAAAC');

-- Create an index and check the index size.
gaussdb=# CREATE INDEX idx_test_c1 ON tbl_test(c1);
gaussdb=# SELECT pg_size_pretty(pg_total_relation_size('idx_test_c1')) AS size;
 size
-------
 64 kB
(1 row)

-- Insert 10,000 data records and then delete the data. It is found that the index becomes larger.
gaussdb=# INSERT INTO tbl_test VALUES (generate_series(1,10000),'test');
gaussdb=# DELETE FROM tbl_test WHERE c2 = 'test';
gaussdb=# SELECT pg_size_pretty(pg_total_relation_size('idx_test_c1')) AS size;
  size
--------
 376 kB
(1 row)

-- After an independent index is rebuilt, the index size is restored to the initial size.
gaussdb=# REINDEX INDEX idx_test_c1;
gaussdb=# SELECT pg_size_pretty(pg_total_relation_size('idx_test_c1')) AS size;
 size
-------
 64 kB
(1 row)

-- Rebuild a single index online.
gaussdb=# REINDEX INDEX CONCURRENTLY idx_test_c1;

-- Rebuild all indexes in the table tbl_test.
gaussdb=# REINDEX TABLE tbl_test;

-- Rebuild all indexes in the table tbl_test online.
gaussdb=# REINDEX TABLE CONCURRENTLY tbl_test;

-- Delete the tbl_test table.
gaussdb=# DROP TABLE tbl_test;
```

Suggestions

- INTERNAL TABLE

  This scenario is used for fault recovery. You are advised not to perform concurrent operations.

- DATABASE

  You are not allowed to reindex databases in transactions.
- SYSTEM

  You are not allowed to reindex system catalogs in transactions.

7.12.17.5 RELEASE SAVEPOINT

Description

RELEASE SAVEPOINT destroys a savepoint previously defined in the current transaction.

Destroying a savepoint makes it unavailable as a rollback point, but it has no other user visible behavior. It does not undo the effects of statements executed after the savepoint was established. To do that, use ROLLBACK TO SAVEPOINT. Destroying a savepoint when it is no longer needed allows the system to recycle some resources earlier than transaction end.

RELEASE SAVEPOINT also destroys all savepoints that were established after the named savepoint was established.

Precautions

- Releasing a savepoint name that was not previously defined will cause an error.
- It is not possible to release a savepoint when the transaction is in an aborted state.
- If multiple savepoints have the same name, only the one that was most recently defined is released.

Syntax

```
RELEASE [ SAVEPOINT ] savepoint_name;
```

Parameters

- **savepoint_name**

  Specifies the name of the savepoint to be destroyed.

Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create a table.
gaussdb=# CREATE TABLE tpcds.table1(a int);

-- Start a transaction.
gaussdb=# START TRANSACTION;

-- Insert data.
gaussdb=# INSERT INTO tpcds.table1 VALUES (3);

-- Establish a savepoint.
gaussdb=# SAVEPOINT my_savepoint;

-- Insert data.
```



```
gaussdb=# INSERT INTO tpcds.table1 VALUES (4);

-- Delete the savepoint.
gaussdb=# RELEASE SAVEPOINT my_savepoint;

-- Commit the transaction.
gaussdb=# COMMIT;

-- Query the table content, which should contain both 3 and 4.
gaussdb=# SELECT * FROM tpcds.table1;

-- Delete the table.
gaussdb=# DROP TABLE tpcds.table1;

-- Delete a schema.
gaussdb=# DROP SCHEMA tpcds CASCADE;
```

Helpful Links

SAVEPOINT and **ROLLBACK TO SAVEPOINT**

7.12.17.6 RENAME TABLE

Description

In the same statement, changing the name of a single table or multiple tables does not affect the stored data.

Precautions

- When the name of a single table is changed, this syntax is equivalent to RENAME in ALTER TABLE.
- Changing the names of multiple tables is equivalent to running **ALTER TABLE** for multiple times. However, changing the names of local and non-local temporary tables at the same time is not supported.

Syntax

```
RENAME { TABLE | TABLES } table_name TO new_table_name [, table_name2 TO new_table_name2, ...];
```

NOTE

If you run this command in a version 5.7 B-compatible database (**sql_compatibility** set to **'B'**, **b_format_version** set to **'5.7'**, and **b_format_dev_version** set to **'s2'**), the following situations may occur:

- If the new table name starts with "#MySQL50#" and is followed by other characters, "#MySQL50#" will be ignored.
- If the old and new table names are the same, no error is reported.

Parameters

- **TABLE | TABLES**

  TABLE and TABLES can be used interchangeably, regardless of the number of tables operated in the statement.

- **table_name TO new_table_name [, table_name2 TO new_table_name2, ...]**

table_name and **table_name2** indicate the names of the tables to be modified.

new_table_name and **new_table_name2** indicate the new table names.

TO is the connection word.

Examples

- Rename a single table.

```
gaussdb=# CREATE TABLE aa(c1 int, c2 int);
gaussdb=# RENAME TABLE aa TO test_alt1;
gaussdb=# DROP TABLE test_alt1;
```

- Rename multiple tables.

```
gaussdb=# CREATE TABLE aa(c1 int, c2 int);
gaussdb=# CREATE TABLE bb(c1 int, c2 int);
gaussdb=# RENAME TABLE aa TO test_alt1, bb TO test_alt2;
gaussdb=# DROP TABLE test_alt1,test_alt2;
```

Helpful Links

ALTER TABLE

7.12.17.7 REPLACE

Description

Inserts data into a table or replaces existing data in a table. If the data to be inserted has the primary key or unique key conflicts with the existing data, the REPLACE statement deletes the existing data and then inserts the new data.

You can use REPLACE in the following ways:

- Replace or insert values. That is, VALUES or VALUE is used to construct a row of records and insert the row into the table.
- Replace or insert the query. One or more rows of records are constructed based on the result set returned by SELECT and inserted into a table.
- Set the value of a specified column. Similar to value insertion, the default value is used for columns that are not specified.

Precautions

- To execute this statement, you must have the DELETE and INSERT permissions on the table.
- If the primary key or unique key does not conflict, you can directly insert the data. If the primary key or unique key conflicts, delete the original data and then insert the new data.
- The return format of the REPLACE operation is **REPLACE 0** *X*, where *X* indicates the number of DELETE and INSERT operations.
- In the REPLACE...SELECT syntax, the number of columns in **select_list** must be the same as the number of columns to be inserted.
- In the REPLACE...SET syntax:
  - If **col_name** has a default value, **SET col_name = col_name + 1** is equivalent to **SET col_name =** *Default value of col_name* **+ 1**.

- If **col_name** has neither default value nor NOT NULL constraint, **SET col_name = col_name + 1** is equivalent to **col_name = NULL**.
- If **col_name** does not have a default value but has a NOT NULL constraint, the data types that support a default value are timestamp, timestamp with time zone, time, time with time zone, interval, tinterval, smalldatetime, date, uuid, name, point, polygon, circle, lseg, box, json, jsonb, xml, varbit, numeric, cidr, inet, macaddr, numrange, int8range, int4range, tsrange, tstzrange, daterange, hash16, hash32, bool, bytea, char, bigint, int, smallint, tinyint, text, raw, blob, clob, float4, float8, abstime, reltime, bpchar, varchar, nvarchar, money, uint1, uint2, uint4, uint8, set, and enum. The default values are described in **Table 7-225**.

Table 7-225 Default values for the data type

Data Type	Default Value of col_name
int8, int4, int2, int1, float8, float4, numeric, uint1, uint2, uint4, and uint8	**0**. If **numeric** specifies a decimal place, the decimal place is displayed. For example, **NUMERIC (10, 3): 0.000**
text, clob, bpchar, varchar, char, nvarchar2, name, blob, raw, and varbit	Empty string.
numrange, int8range, int4range, tsrange, tstzrange, and daterange	empty
bytea	\x
money	$0.00
json, jsonb, and xml	'null'
macaddr	00:00:00:00:00:00
inet	0.0.0.0
cidr	0.0.0.0/32
point	(0,0)
lseg	[(0,0),(0,0)]
box	(0,0),(0,0)
path	((0,0))
polygon	((0,0))
circle	<(0,0),0>
uuid	00000000-0000-0000-0000-000000000000
hash16	0000000000000000

Data Type	Default Value of col_name
hash32	00000000000000000000000000000000
bool	f
abstime	abstime '1970-01-01 00:00:00'
reltime	reltime '00:00:00'
interval	interval '00:00:00'
tinterval	tinterval(abstime '1970-01-01 00:00:00', abstime '1970-01-01 00:00:00')
timestamp	timestamp '1970-01-01 00:00:00'
timestamp with time zone	timestamptz '1970-01-01 00:00:00'
date	date '1970-01-01'
time	time '00:00:00'
time with time zone	timetz '00:00:00'
smalldatetime	smalldatetime '1970-01-01 00:00:00'

NOTE

- The default value of the enumerated type (ENUM) is the first element. If the first element does not exist, **NULL** is returned.
- The SET data type is supported only when **sql_compatibility** is set to **'B'**. The default value is an empty string.
- When **sql_compatibility** is set to **'A'**, an empty string of the text, clob, blob, raw, bytea, varchar, nvarchar2, bpchar, char, name, byteawithoutorderwithqualcol, or byteawithoutordercol type is equivalent to **NULL**. If a column has a NOT NULL constraint, an error is reported when a reference column is used to insert data into a table.
- When **sql_compatibility** is set to **'B'**, if **b_format_version** is set to **'5.7'**, **b_format_dev_version** is set to **'s1'**, and **sql_mode** does not contain **strict_tans_tables**, **only_full_group_by**, **no_zero_in_date**, **no_zero_date**, or **error_for_division_by_zero**, then the default values of **timestamp** and **datetime** are **0000-00-00 00:00:00**, and the default value of **DATE** is **0000-00-00** under the NOT NULL constraint.
- The uint1, uint2, uint4, and uint8 data types are supported only when **sql_compatibility** is set to **'B'**.
- If the default value of the function expression can be calculated during parsing, the default value is the calculated constant. Otherwise, the value is **NULL**.
- In other scenarios, the default value is **NULL**.

- In the REPLACE... SET syntax, the current **col_name** value depends on the previous **col_name** value. If there is no previous **col_name** value, the default

value is used. For example, in the **SET f1 = f1 + 1, f2 = f1** scenario, **f1** is equal to the default value of **f1** (assumed to **0**) plus 1, and **f2** is equal to the calculated value of **f1**, that is **1**.

- Triggers are supported. The execution sequence of triggers is determined by the actual execution process.
  - Executing INSERT will trigger the BEFORE INSERT and AFTER INSERT triggers.
  - Executing DELETE will trigger the BEFORE DELETE and AFTER DELETE triggers.
- The unique constraint or primary key of DEFERRABLE is not supported.
- If a table has multiple unique constraints and the inserted data violates multiple unique constraints, all the data that violates the constraints is deleted and new data is inserted. In this scenario, data may be deleted by mistake. Therefore, exercise caution when performing this operation.
- If multiple rows are inserted and these rows have unique constraint conflicts with data in the same row in the table, the REPLACE operation is performed in sequence.
- In **SET col_name = col_name + 1**, the length of **col_name** cannot exceed 1. For example, in **SET col_name = table_name.col_name + 1**, the length of **table_name.col_name** is 2. Formats such as B.A, C.B.A, and D.C.B.A are not supported.
- When comparing floating-point data, note that precision loss may occur.
- If a row-level security policy is created on a table, REPLACE INTO is not supported.
- Foreign tables are not supported.
- Encrypted tables are not supported.
- Memory tables are not supported.

Syntax

- Replace or insert values.

```
REPLACE [ INTO ] table_name
    [ PARTITION ( partition_name [, ... ] ) ]
    [ ( col_name [, ... ] ) ]
    { VALUES | VALUE } ( value [, ... ] ) [, ... ];
```

- Replace or insert the query.

```
REPLACE [ INTO ] table_name
    [ PARTITION ( partition_name [, ... ] ) ]
    [ ( col_name [, ... ]) ]
    query;
```

- Set the value of a specified column.

```
REPLACE [ INTO ] table_name
    [ PARTITION ( partition_name [, ... ] ) ]
    SET col_name = value [, ... ];
```

Parameters

- **table_name**

  Specifies the name of the target table where data will be inserted.

  Value range: an existing table name.

- **col_name**

Specifies the name of a column in a table.

- The column name can be qualified with a subcolumn name or array index, if needed.
- Each column not present in the column list will be filled with a default value, either its declared default value or **NULL** if there is none. Inserting data into only some columns of a composite type leaves the other columns **NULL**.
- The target column names (specified by **col_name**) can be listed in any order. If no list of column names is given at all, the default is all the columns of the table in their declared order.
- The target columns are the first *N* column names, if there are only *N* columns provided by the VALUE clause and QUERY.
- The values provided by the VALUE clause and QUERY are joined with the corresponding columns from left to right in the table.

Value range: an existing column.

- **PARTITION (partition_name [, ...])**

  Inserts data to a specified partition. **partition_name** indicates the partition name.

  If the value of the VALUE clause is inconsistent with that of the specified partition, an exception is displayed.

- **value**

  Specifies the value to be inserted. The value format is as follows:

  ```
  { expression | DEFAULT }
  ```

  a. **expression** indicates that a valid expression or value is assigned to the corresponding column.

     If single-quotation marks are inserted in a column, the single-quotation marks need to be used for escape.

     If the expression for any column is not of the correct data type, automatic type conversion will be attempted. If the attempt fails, data insertion fails, and the system returns an error message.

  b. **DEFAULT** indicates the default value of the corresponding column name. The value is **NULL** if no default value is assigned to it.

- **query**

  Specifies a query statement (SELECT statement) whose result is used as the data to be inserted.

Examples

```
-- Create a table.
gaussdb=# CREATE TABLE test(f1 int primary key, f2 int, f3 int);

-- Insert data.
gaussdb=# INSERT INTO test VALUES(1, 1, 1), (2, 2, 2), (3, 3, 3);
INSERT 0 3

-- Replace or insert values.
gaussdb=# REPLACE INTO test VALUES(1, 11, 11);
REPLACE 0 2

-- Query the result.
gaussdb=# SELECT * FROM test WHERE f1 = 1;
```



```
 f1 | f2 | f3
----+----+----
  1 | 11 | 11
(1 row)

-- Replace or insert the query.
gaussdb=# REPLACE INTO test SELECT 2, 22, 22;
REPLACE 0 2

-- Query the result.
gaussdb=# SELECT * FROM test WHERE f1 = 2;
 f1 | f2 | f3
----+----+----
  2 | 22 | 22
(1 row)

-- Replace or insert the specified column.
gaussdb=# REPLACE INTO test SET f1 = f1 + 3, f2 = f1 * 10 + 3, f3 = f2;
REPLACE 0 2

-- Query the result.
gaussdb=# SELECT * FROM test WHERE f1 = 3;
 f1 | f2 | f3
----+----+----
  3 | 33 | 33
(1 row)

-- Delete the table.
gaussdb=# DROP TABLE test;
```

7.12.17.8 RESET

Description

RESET restores GUC parameters to their default values. The default values are parameter default values complied in the **gaussdb.conf** configuration file.

RESET is an alternative spelling for:

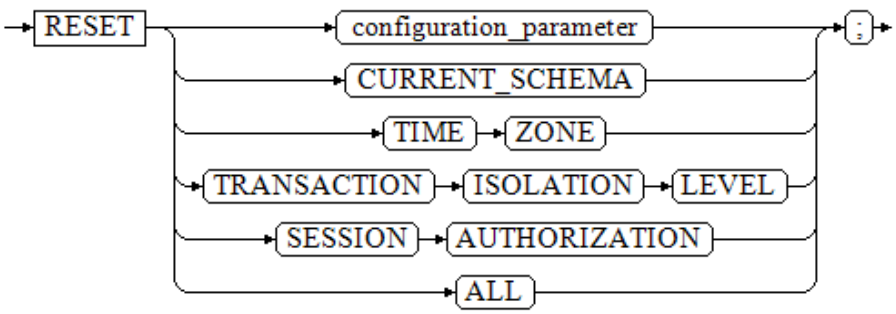
```
SET configuration_parameter TO DEFAULT;
```

Precautions

RESET and SET have the same transaction behavior. Their impact will be rolled back.

Syntax

```
RESET {configuration_parameter | CURRENT_SCHEMA | TIME ZONE | TRANSACTION ISOLATION LEVEL |
SESSION AUTHORIZATION | ALL };
```



Parameters

- **configuration_parameter**

  Specifies the name of a GUC parameter.

  Value range: GUC parameters. You can view them by running the **SHOW ALL** statement.

- **CURRENT_SCHEMA**

  Specifies the current schema.

- **TIME ZONE**

  Specifies the time zone.

- **TRANSACTION ISOLATION LEVEL**

  Specifies the transaction isolation level.

- **SESSION AUTHORIZATION**

  Specifies the session authorization.

- **ALL**

  Resets all GUC parameters to default values.

Examples

```
-- Set the time zone to Italy.
gaussdb=# SET timezone TO 'Europe/Rome';
-- View the current time zone.
gaussdb=# SHOW timezone;
  TimeZone
-------------
 Europe/Rome
(1 row)

-- Set the time zone to the default value.
gaussdb=# RESET timezone;
-- View the current time zone.
gaussdb=# SHOW timezone;
 TimeZone
----------
 PRC
(1 row)

-- The preceding SQL statement is equivalent to the following two SQL statements. Set the time zone to the default value.
gaussdb=# SET timezone TO DEFAULT;
gaussdb=# ALTER SESSION SET timezone to DEFAULT;
```

Helpful Links

SET and **SHOW**

7.12.17.9 REVOKE

Description

Revokes permissions from one or more roles.

Precautions

If a non-owner user of an object attempts to REVOKE permission on the object, the statement is executed based on the following rules:

- If the user has no permissions whatsoever on the object, the statement will fail outright.
- If an authorized user has some permissions, only the permissions with authorization options are revoked.
- If the authorized user does not have the authorization option, the REVOKE ALL PRIVILEGES form will issue an error message. For other forms of statements, if the permission specified in the statement does not have the corresponding authorization option, the statement will issue a warning.

Syntax

- Revoke the permission on a specified table or view.

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | ALTER | DROP | COMMENT |
INDEX | VACUUM }[, ...]
    | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
        | ALL TABLES IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified field in a table.

```
REVOKE [ GRANT OPTION FOR ]
    { {{ SELECT | INSERT | UPDATE | REFERENCES | COMMENT } ( column_name [, ...] )}[, ...]
    | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
    ON [ TABLE ] table_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified sequence. The **LARGE** field is optional. The recycling statement does not distinguish whether the sequence is LARGE.

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | UPDATE | ALTER | DROP | COMMENT }[, ...]
    | ALL [ PRIVILEGES ] }
    ON { [ [ LARGE ] SEQUENCE ] sequence_name [, ...]
        | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified database.

```
REVOKE [ GRANT OPTION FOR ]
    { { CREATE | CONNECT | TEMPORARY | TEMP | ALTER | DROP | COMMENT } [, ...]
    | ALL [ PRIVILEGES ] }
    ON DATABASE database_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified domain.

```
REVOKE [ GRANT OPTION FOR ]
    { USAGE | ALL [ PRIVILEGES ] }
    ON DOMAIN domain_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified CMK.

```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | DROP } [, ...] | ALL [PRIVILEGES] }
    ON CLIENT_MASTER_KEYS client_master_keys_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```


- Revoke the permission on a specified CEK.

```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | DROP } [, ...] | ALL [PRIVILEGES]}
    ON COLUMN_ENCRYPTION_KEYS column_encryption_keys_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified directory.

```
REVOKE [ GRANT OPTION FOR ]
    { { READ | WRITE | ALTER | DROP } [, ...] | ALL [ PRIVILEGES ] }
    ON DIRECTORY directory_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified external data source.

```
REVOKE [ GRANT OPTION FOR ]
  { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN DATA WRAPPER fdw_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified external server.

```
REVOKE [ GRANT OPTION FOR ]
    { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON FOREIGN SERVER server_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified function.

```
REVOKE [ GRANT OPTION FOR ]
    { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON { FUNCTION {function_name ( [ {[ argmode ] [ arg_name ] arg_type} [, ...] ] )} [, ...]
      | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified stored procedure.

```
REVOKE [ GRANT OPTION FOR ]
    { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON { PROCEDURE {proc_name ( [ {[ argmode ] [ arg_name ] arg_type} [, ...] ] )} [, ...]
      | ALL PROCEDURE IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified procedural language.

```
REVOKE [ GRANT OPTION FOR ]
  { USAGE | ALL [ PRIVILEGES ] }
    ON LANGUAGE lang_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified large object.

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | UPDATE } [, ...] | ALL [ PRIVILEGES ] }
    ON LARGE OBJECT loid [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified schema.

```
REVOKE [ GRANT OPTION FOR ]
    { { CREATE | USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON SCHEMA schema_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```

- Revoke the permission on a specified tablespace.

```
REVOKE [ GRANT OPTION FOR ]
    { { CREATE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
    ON TABLESPACE tablespace_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ];
```


- Revoke the permission on a specified type.
```
REVOKE [ GRANT OPTION FOR ]
  { { USAGE | ALTER | DROP | COMMENT } [, ...] | ALL [ PRIVILEGES ] }
   ON TYPE type_name [, ...]
   FROM { [ GROUP ] role_name | PUBLIC } [, ...]
   [ CASCADE | RESTRICT ];
```
- Revoke the permission on a package object.
```
REVOKE [ GRANT OPTION FOR ]
  { { EXECUTE | ALTER | DROP | COMMENT } [, ...] | ALL [PRIVILEGES] }
  ON PACKAGE package_name [, ...]
  FROM {[GROUP] role_name | PUBLIC} [, ...]
  [ CASCADE | RESTRICT ];
```
- Revoke permissions from a role.
```
REVOKE [ ADMIN OPTION FOR ]
   role_name [, ...] FROM role_name [, ...]
   [ CASCADE | RESTRICT ];
```
- Revoke the sysadmin permission on a role.
```
REVOKE ALL { PRIVILEGES | PRIVILEGE } FROM role_name;
```
- Revoke the ANY permissions.
```
REVOKE [ ADMIN OPTION FOR ]
 { CREATE ANY TABLE | ALTER ANY TABLE | DROP ANY TABLE | SELECT ANY TABLE | INSERT ANY
TABLE | UPDATE ANY TABLE |
  DELETE ANY TABLE | CREATE ANY SEQUENCE | CREATE ANY INDEX | CREATE ANY FUNCTION |
EXECUTE ANY FUNCTION |
  CREATE ANY PACKAGE | EXECUTE ANY PACKAGE | CREATE ANY TYPE | ALTER ANY TYPE | DROP ANY
TYPE | ALTER ANY SEQUENCE | DROP ANY SEQUENCE |
  SELECT ANY SEQUENCE | ALTER ANY INDEX | DROP ANY INDEX | CREATE ANY SYNONYM | DROP
ANY SYNONYM | CREATE ANY TRIGGER | ALTER ANY TRIGGER |  DROP ANY TRIGGER
} [, ...]
  FROM [ GROUP ] role_name [, ...];
```
- Revoke the permission on database links.
```
REVOKE { CREATE | ALTER | DROP } [PUBLIC] DATABASE LINK FROM role_name;
```

  NOTE

  For details about database links, see **DATABASE LINK**.

- Revoke the permission on PUBLIC synonyms.
```
REVOKE { CREATE | DROP } PUBLIC SYNONYM FROM role_name;
```

  Built-in roles (gs_role_public_synonym_create and gs_role_public_synonym_drop) can also be used to revoke the permission on PUBLIC synonyms.

  – Revoke the permission to create PUBLIC synonyms.
```
REVOKE gs_role_public_synonym_create FROM role_name;
```
  – Revoke the permission to delete PUBLIC synonyms.
```
REVOKE gs_role_public_synonym_drop FROM role_name;
```

Parameters

The keyword PUBLIC indicates an implicitly defined group that has all roles.

For details about permission types and parameters, see **Parameters** in section "GRANT".

Permissions of a role include the permissions directly granted to the role, permissions inherited from the parent role, and permissions granted to PUBLIC users. Therefore, revoking the SELECT permission on an object from PUBLIC users does not necessarily mean that the SELECT permission on the object has been revoked from all roles, because the SELECT permission directly granted to roles

and inherited from parent roles remains. Similarly, if the SELECT permission is revoked from a user but is not revoked from PUBLIC users, the user can still run the SELECT statement.

If GRANT OPTION FOR is specified, the permission cannot be granted to others, but permission itself is not revoked.

If user A holds the UPDATE permission on a table and the **WITH GRANT OPTION** option and has granted them to user B, the permission that user B holds is called dependent permission. When user A's permission or grant option is revoked, CASCADE must be declared to revoke all dependent permissions.

A user can only revoke permissions that were granted directly by that user. For example, if user A has granted permission with grant option (**WITH ADMIN OPTION**) to user B, and user B has in turn granted it to user C, then user A cannot revoke the permission directly from C. However, user A can revoke the grant option held by user B and use CASCADE. In this way, the permission of user C is automatically revoked. For another example, if both user A and user B have granted the same permission to C, A can revoke his own grant but not B's grant, so C will still effectively have the permission.

If the role executing REVOKE holds permissions indirectly by using more than one role membership path, it is unspecified which containing role will be used to execute the statement. In such cases, you are advised to use SET ROLE to become the specific role, and then execute REVOKE. Failure to do so may lead to deleting permissions not intended to delete, or not deleting any permissions at all.

Examples

- Revoke the permission of role **jerry** from user **tom**.
  ```
  gaussdb=# REVOKE jerry FROM tom;
  REVOKE ROLE
  ```
- Revoke the SELECT permission for the **t1** table in schema **jerry** from user **tom**.
  ```
  gaussdb=# REVOKE SELECT ON TABLE jerry.t1 FROM tom;
  REVOKE
  ```
- Revoke the EXECUTE permission for the fun1 function in schema **jerry** from user **tom**.
  ```
  gaussdb=# REVOKE EXECUTE ON FUNCTION jerry.fun1() FROM tom;
  REVOKE
  ```
- Revoke the CONNECT permission on database **DB1** from user **tom**.
  ```
  gaussdb=# REVOKE CONNECT ON database DB1 FROM tom;
  REVOKE
  ```

For more examples, see **Examples** in section "GRANT."

Helpful Links

GRANT

7.12.17.10 ROLLBACK

Description

Rolls back the current transaction and backs out all updates in the transaction.

If a fault occurs during the running of a transaction, the transaction cannot be executed. The system cancels all completed operations on the database in the

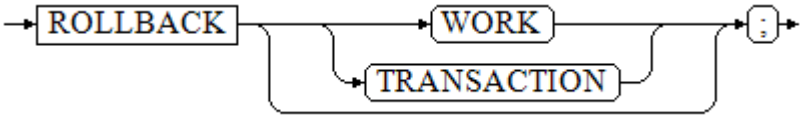
transaction, and the database status returns to the time when the transaction starts.

Precautions

If a ROLLBACK statement is executed out of a transaction, no error occurs, but a notice is displayed.

Syntax

```
ROLLBACK [ WORK | TRANSACTION ];
```



Parameters

WORK | TRANSACTION

Specifies the optional keyword that more clearly illustrates the syntax.

Examples

```
-- Create a table.
gaussdb=# CREATE TABLE test (id int, name text);

-- Start a transaction.
gaussdb=# START TRANSACTION;

-- Alter the table structure.
gaussdb=# ALTER TABLE test ADD COLUMN score int;

-- View the table structure.
gaussdb=# \d test;
     Table "public.test"
 Column |  Type
--------+---------
 id     | integer
 name   | text
 score  | integer

-- Perform rollback.
gaussdb=# ROLLBACK;

-- The table structure is restored to the initial state.
gaussdb=# \d test;
     Table "public.test"
 Column |  Type
--------+---------
 id     | integer
 name   | text
```

Helpful Links

COMMIT | END

7.12.17.11 ROLLBACK PREPARED

Description

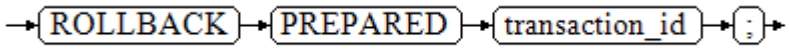
Cancels a transaction ready for two-phase commit.

Precautions

- The function is only available in maintenance mode (when the GUC parameter **xc_maintenance_mode** is set to **on**). Exercise caution when enabling the mode. It is used by maintenance engineers for troubleshooting. Common users should not use the mode.
- Only the user who initiates a transaction or a system administrator can roll back the transaction.
- The transaction function is maintained automatically by the database, and should be not visible to users.

Syntax

```
ROLLBACK PREPARED transaction_id ;
```



Parameters

transaction_id

Specifies the identifier of the transaction to be committed. The identifier must be different from those for current prepared transactions.

Examples

```
-- Start.
gaussdb=# BEGIN;
BEGIN

-- Prepare a transaction whose identifier is trans_test.
gaussdb=# PREPARE TRANSACTION 'trans_test';
PREPARE TRANSACTION

-- Cancel the transaction whose identifier is trans_test.
gaussdb=# ROLLBACK PREPARED 'trans_test';
ROLLBACK PREPARED
```

Helpful Links

COMMIT PREPARED and **PREPARE TRANSACTION**.

7.12.17.12 ROLLBACK TO SAVEPOINT

Description

ROLLBACK TO SAVEPOINT rolls back to a savepoint. It implicitly destroys all savepoints that were established after the named savepoint.

Rolls back all statements that were executed after the savepoint was established. The savepoint remains valid and can be rolled back to again later, if needed.

Precautions

- Specifying a savepoint name that has not been established is an error.
- Cursors have somewhat non-transactional behavior with respect to savepoints. Any cursor that is opened inside a savepoint will be closed when the savepoint is rolled back. If a previously opened cursor is affected by a FETCH or MOVE statement inside a savepoint that is later rolled back, the cursor remains at the position that FETCH left it pointing to (that is, the cursor motion caused by FETCH is not rolled back). Closing a cursor is not undone by rolling back, either. A cursor whose execution causes a transaction to abort is put in a cannot-execute state, so while the transaction can be restored using ROLLBACK TO SAVEPOINT, the cursor can no longer be used.
- Use ROLLBACK TO SAVEPOINT to roll back to a savepoint. Use RELEASE SAVEPOINT to destroy a savepoint but keep the effects of the statements executed after the savepoint was established.

Syntax

```
ROLLBACK [ WORK | TRANSACTION ] TO [ SAVEPOINT ] savepoint_name;
```

Parameters

- **savepoint_name**

  Rolls back to a savepoint.

Examples

```
-- Start a transaction.
gaussdb=# START TRANSACTION;

-- Set the savepoint my_savepoint.
gaussdb=# SAVEPOINT my_savepoint;

-- Roll back to the savepoint my_savepoint.
gaussdb=# ROLLBACK TO SAVEPOINT my_savepoint;

-- Cursor positions are not affected by savepoint rollback.
gaussdb=# DECLARE foo CURSOR FOR SELECT 1 UNION SELECT 2;

-- Set the savepoint foo.
gaussdb=# SAVEPOINT foo;

-- Read a data record, which is the first one.
gaussdb=# FETCH 1 FROM foo;
 ?column?
----------
        1
-- Roll back to the savepoint foo.
gaussdb=# ROLLBACK TO SAVEPOINT foo;

-- Read a data record, which is the second one.
gaussdb=# FETCH 1 FROM foo;
 ?column?
----------
        2
-- Release a savepoint.
gaussdb=# RELEASE SAVEPOINT my_savepoint;
```



```
-- End the transaction.
gaussdb=# COMMIT;
```

Helpful Links

SAVEPOINT and **RELEASE SAVEPOINT**

7.12.18 S

7.12.18.1 SAVEPOINT

Description

SAVEPOINT establishes a new savepoint in the current transaction.

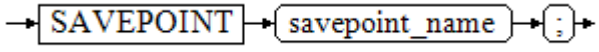
A savepoint is a special mark inside a transaction. It allows all statements that are executed after it was established to be rolled back, restoring the transaction state to what it was at the time of the savepoint.

Precautions

- Use ROLLBACK TO SAVEPOINT to roll back to a savepoint. Use RELEASE SAVEPOINT to destroy a savepoint but keep the effects of the statements executed after the savepoint was established.
- Savepoints can only be established when inside a transaction block. Multiple savepoints can be defined in a transaction.
- In the case of an unexpected termination of a thread or process caused by a node or connection failure, or of an error caused by the inconsistency between source and destination table structures in a COPY FROM operation, the transaction cannot be rolled back to the established savepoint. Instead, the entire transaction will be rolled back.
- According to the SQL standard, when a savepoint with the same name is created, the previous savepoint with the same name is automatically deleted. In GaussDB, the old savepoint is retained, but only the latest one is used during rollback or release. If the latest savepoint is released, the previous savepoint will again become accessible to ROLLBACK TO SAVEPOINT and RELEASE SAVEPOINT. In addition, SAVEPOINT fully complies with the SQL standard.

Syntax

```
SAVEPOINT savepoint_name;
```



Parameters

savepoint_name

Specifies the name of the new savepoint.

NOTICE

When using SAVEPOINT, you are advised to release SAVEPOINT promptly to avoid too many nested subtransactions. It is recommended that the number of nested subtransactions be less than or equal to 10,000. If the number of nested subtransactions is too large, the current transaction performance may deteriorate.

Examples

```
-- Create a table.
gaussdb=# CREATE TABLE table1(a int);

-- Start a transaction.
gaussdb=# START TRANSACTION;

-- Insert data.
gaussdb=# INSERT INTO table1 VALUES (1);

-- Create a savepoint.
gaussdb=# SAVEPOINT my_savepoint;

-- Insert data.
gaussdb=# INSERT INTO table1 VALUES (2);

-- Roll back the savepoint.
gaussdb=# ROLLBACK TO SAVEPOINT my_savepoint;

-- Insert data.
gaussdb=# INSERT INTO table1 VALUES (3);

-- Commit the transaction.
gaussdb=# COMMIT;

-- Query the content of the table. You can see 1 and 3 at the same time, but cannot see 2 because 2 is
rolled back.
gaussdb=# SELECT * FROM table1;

-- Delete the table.
gaussdb=# DROP TABLE table1;

-- Create a table.
gaussdb=# CREATE TABLE table2(a int);

-- Start a transaction.
gaussdb=# START TRANSACTION;

-- Insert data.
gaussdb=# INSERT INTO table2 VALUES (3);

-- Create a savepoint.
gaussdb=# SAVEPOINT my_savepoint;

-- Insert data.
gaussdb=# INSERT INTO table2 VALUES (4);

-- Roll back the savepoint.
gaussdb=# RELEASE SAVEPOINT my_savepoint;

-- Commit the transaction.
gaussdb=# COMMIT;

-- Query the table content. You can see 3 and 4 at the same time.
gaussdb=# SELECT * FROM table2;

-- Delete the table.
gaussdb=# DROP TABLE table2;
```


Helpful Links

RELEASE SAVEPOINT and **ROLLBACK TO SAVEPOINT**

7.12.18.2 SECURITY LABEL ON

Function

Applies, updates, or cancels a security label.

Precautions

An initial user, a user with the SYSADMIN permission, or a user who inherits the gs_role_seclabel permission of the built-in role can update or cancel security labels.

Syntax

```
SECURITY LABEL ON { ROLE | USER | TABLE | COLUMN } objname IS {'label_name' | NULL};
```

Parameter Description

- **objname**
  - For **ROLE** and **USER**, **objname** indicates the user/role name.
  - For **TABLE**, **objname** indicates the table name, which can be prefixed with a schema name.
  - For **COLUMN**, **objname** indicates the name in the format of "table name.column name", which can be prefixed with a schema name.
- **label_name**

  Specifies the security label name.
- **NULL**

  Specifies that the security label is canceled.

Examples

```
-- Create a security label.
gaussdb=# CREATE SECURITY LABEL sec_label 'L1:G4';
-- Create a table.
gaussdb=# CREATE TABLE tbl(c1 int, c2 int);
-- Create a user.
gaussdb=# CREATE USER bob WITH PASSWORD '********';
-- Apply a security label to the user.
gaussdb=# SECURITY LABEL ON ROLE bob IS 'sec_label';
-- Apply a security label to the table.
gaussdb=# SECURITY LABEL ON TABLE tbl IS 'sec_label';
-- Apply a security label to a column of the table.
gaussdb=# SECURITY LABEL ON COLUMN tbl.c1 IS 'sec_label';
-- Cancel the security label of the user.
gaussdb=# SECURITY LABEL ON ROLE bob IS NULL;
-- Cancel the security label of the table.
gaussdb=# SECURITY LABEL ON TABLE tbl IS NULL;
-- Cancel the security label of the column of the table.
gaussdb=# SECURITY LABEL ON COLUMN tbl.c1 IS NULL;
-- Delete the existing security label sec_label.
gaussdb=# DROP SECURITY LABEL sec_label;
-- Delete table tb1.
gaussdb=# DROP TABLE tbl;
```



```
-- Delete user bob.
gaussdb=# DROP USER bob;
```

Helpful Links

CREATE SECURITY LABEL and **DROP SECURITY LABEL**

7.12.18.3 SELECT

Description

Retrieves data from a table or view.

Serving as an overlaid filter on database tables, the SELECT statement uses SQL keywords to filter data tables and extract the required data.

Precautions

- The owner of the table, users granted with the SELECT permission on the table, and users granted with the SELECT ANY TABLE permission can read data in the table or view. When the separation of duties is disabled, system administrators have this permission by default.
- You must have the SELECT permission on each field used in the SELECT statement.
- Using FOR UPDATE, FOR NO KEY UPDATE, FOR SHARE, or FOR KEY SHARE also requires the UPDATE permission in addition to the SELECT permission.

Syntax

- Query data.

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [/*+ plan_hint */] [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
{ * | {expression [ [ AS ] output_name ]} [, ...] }
[ into_option ]
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ [ START WITH condition ] CONNECT BY [NOCYCLE] condition [ ORDER SIBLINGS BY expression ] ]
[ GROUP BY grouping_element [, ...] ]
[ HAVING condition [, ...] ]
[ WINDOW {window_name AS ( window_definition )} [, ...] ]
[ { UNION | INTERSECT | EXCEPT | MINUS } [ ALL | DISTINCT ] select ]
[ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS { FIRST | LAST } ]} [, ...] ]
[ LIMIT { [offset,] count | ALL } ]
[ OFFSET start [ ROW | ROWS ] ]
[ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
[ into_option ]
[ {FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE } [ OF table_name [, ...] ] [ NOWAIT | WAIT n | SKIP LOCKED ]}[...] ]
[into_option];
```


NOTE

In condition and expression, you can use the aliases of expressions in **targetlist** in compliance with the following rules:

- Reference only within the same level.
- Only reference aliases in **targetlist**.
- Reference a prior expression in a subsequent expression.
- The **volatile** function cannot be used.
- The **Window** function cannot be used.
- Aliases cannot be referenced in the **JOIN ON** condition.
- An error is reported if the target list contains multiple aliases to be referenced.

NOTICE

In the scenario where the SELECT statement plan is cached, the WHERE IN candidate subset cannot be too large. It is recommended that the number of conditions be less than or equal to 100 to prevent high dynamic memory.

- If the WHERE IN candidate subset is too large, the memory usage of the generated plan increases.
- If the WHERE IN subsets constructed by concatenated SQL statements are different, the SQL template of the cache plan cannot be reused. A large number of different plans are generated, and the plans cannot be shared. As a result, a large amount of memory is occupied.

- The subquery **with_query** is as follows:

```
with_query_name [ ( column_name [, ...] ) ]
    AS [ [ NOT ] MATERIALIZED ] ( {select | values | insert | update | delete} )
```

- The INTO clause is as follows:

```
into_option: {
        INTO var_name [, var_name] ...
    | INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
    | INTO DUMPFILE 'file_name'
}
export_options: {
[FIELDS
    [TERMINATED BY 'string']
    [[OPTIONALLY] ENCLOSED BY 'char']
    [ESCAPED BY 'char' ]
]
[LINES
    [STARTING BY 'string']
    [TERMINATED BY 'string']
 ]
}
```

- The specified query source **from_item** is as follows:

```
{[ ONLY ] table_name [ * ] [ partition_clause ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
[ TABLESAMPLE sampling_method ( argument [, ...] ) [ REPEATABLE ( seed ) ] ]
[ TIMECAPSULE {TIMESTAMP | CSN} expression ]
|( select ) [ AS ] alias [ ( column_alias [, ...] ) ]
|with_query_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
|function_name ( [ argument [, ...] ] ) [ AS ] alias [ ( column_alias [, ...] | column_definition [, ...] ) ]
|function_name ( [ argument [, ...] ] ) AS ( column_definition [, ...] )
|xmltable_clause
|from_item unpivot_clause
```



```
|from_item pivot_clause
|from_item [ NATURAL ] join_type from_item [ ON join_condition | USING ( join_column [, ...] ) ]}
```

- The **group** clause is as follows:

```
| expression
| ( expression [, ...] )
| ROLLUP ( { expression | ( expression [, ...] ) } [, ...] )
| CUBE ( { expression | ( expression [, ...] ) } [, ...] )
| GROUPING SETS ( grouping_element [, ...] )
```

- The specified partition **partition_clause** in **from_item** is as follows:

```
PARTITION { ( { partition_name | subpartition_name } [, ...] ) | FOR (  partition_value [, ...] ) } |
SUBPARTITION { ( subpartition_name ) | FOR (  subpartition_value [, ...] )}
```

  NOTE

  - The specified partition applies only to partitioned tables.
  - If PARTITION specifies multiple partition names, level-1 and level-2 partition names can coexist and can be the same. The union set of the partition range is used.

- The sorting order **nlssort_expression_clause** is as follows:

```
NLSSORT ( column_name, ' NLS_SORT = { SCHINESE_PINYIN_M | generic_m_ci } ' )
```

  The second parameter can be **generic_m_ci**, which supports only the case-insensitive order for English characters.

- Simplified query syntax, equivalent to SELECT * FROM table_name.

```
TABLE { ONLY {(table_name)| table_name} | table_name [ * ]};
```

Parameters

- **WITH [RECURSIVE] with_query [, ...]**

  Specifies one or more subqueries that can be referenced by name in the main query, which is equivalent to a temporary table. This subquery statement structure is called the common table expression (CTE) structure. When this structure is used, the execution plan contains the CTE SCAN content.

  If **RECURSIVE** is specified, it allows a **SELECT** subquery to reference itself by name.

  The detailed format of **with_query** is as follows: **with_query_name [(column_name [, ...])] AS [[NOT] MATERIALIZED] ({select | values | insert | update | delete})**

  - **with_query_name** specifies the name of the result set generated by a subquery. Such names can be used to access the result sets of subqueries in a query.
  - **column_name** specifies the column name displayed in the subquery result set.
  - Each subquery can be a SELECT, VALUES, INSERT, UPDATE, or DELETE statement.
  - RECURSIVE can appear only after WITH. In the case of multiple CTEs, you only need to declare RECURSIVE at the first CTE.
  - You can use **MATERIALIZED** or **NOT MATERIALIZED** to modify the CTE.
    - If **MATERIALIZED** is specified, the WITH query will be materialized, and a copy of the subquery result set is generated. The copy is directly queried at the reference point. Therefore, the WITH subquery cannot be jointly optimized with the SELECT statement trunk (for example, predicate pushdown and equivalence class transfer). In this scenario, you can use **NOT MATERIALIZED** for modification. If the

WITH query can be executed as a subquery inline, the preceding optimization can be performed.

  - If the user does not explicitly declare the materialized attribute, comply with the following rules: If the CTE is referenced only once in the SELECT statement trunk to which it belongs and semantically supports inline execution, it will be rewritten as subquery inline execution. Otherwise, the materialized execution will be performed in CTE Scan mode.

- **plan_hint** clause

  Follows the SELECT keyword in the **/*+**<*Plan hint*>** */** format. It is used to optimize the plan of a SELECT statement block. For details, see **Hint-based Tuning**. In each statement, only the first **/*+** *plan_hint* ***/** comment block takes effect as a hint. Multiple hints can be written.

- **ALL**

  Specifies that all rows that meet the conditions are returned. This is the default behavior and can be omitted.

- **DISTINCT [ON (expression [, ...])]**

  Removes all duplicate rows from the **SELECT** result set so one row is kept from each group of duplicates.

  Only the first row in the set of rows that have the same result calculated on the given expression **ON (expression [, ...])** is retained.

  NOTICE

  DISTINCT ON expression is explained with the same rule of **ORDER BY**. Unless you use **ORDER BY** to guarantee that the required row appears first, you cannot know what the first row is.

- **SELECT list**

  Specifies the name of a column in the table to be queried. The value can be a part of the column name or all of the column names. The wildcard (*) is used to represent the column name.

  You may use the **AS output_name** clause to give an alias for an output column. The alias is used for the displaying of the output column. The name, value, and type keywords can be used as column aliases.

  Column names can be expressed in the following formats:

  - Manually input column names which are separated by commas (,)
  - Columns computed in the **FROM** clause.

- **INTO** clause

  Exports the result of SELECT to a specified user-defined variable or file.

  - var_name

    User-defined variable name. For details, see **var_name** in **SET**.

  - OUTFILE

    - **CHARACTER SET** specifies the encoding format.

- **FIELDS** specifies the attribute of each column.
  - **TERMINATED** specifies the separator.
  - **[OPTIONALLY] ENCLOSED** specifies the quotation mark. When **OPTIONALLY** is specified, it takes effect only for the string data type.
  - **ESCAPED** specifies the escape character.
- **LINES** specifies the row attribute.
  - **STARTING** specifies the beginning of a row.
  - **TERMINATED** specifies the end of a row.

– DUMPFILE

Exports a single row of data without separators or linefeed characters to a file.

– file_name

Specifies the absolute path of a file.

```
Three positions of into_option are as follows:
-- Create custom variables.
gaussdb=# CREATE DATABASE user_var dbcompatibility 'b';
gaussdb=# \c user_var
user_var=# SET b_format_behavior_compat_options = enable_set_variables;
user_var=# SET @my_var := 1;
-- Create table t.
user_var=# CREATE TABLE t (a int, b text);
user_var=# INSERT INTO t values(1,'a');
-- Before the FROM clause
user_var=# SELECT a INTO @my_var FROM t;
-- Before the LOCK clause
user_var=# SELECT a FROM t INTO @my_var FOR UPDATE;
-- At the end of the SELECT statement
user_var=# SELECT a FROM t FOR UPDATE INTO @my_var;

Export to a file.
user_var=# SELECT * FROM t;
 a | b
---+---
 1 | a
(1 row)
-- Export data to the outfile file. (Change the absolute path of the outfile file as required.)
user_var=# SELECT * FROM t INTO OUTFILE '/home/gaussdb/t.txt' FIELDS TERMINATED BY '~'
ENCLOSED BY 't' ESCAPED BY '^' LINES STARTING BY '$' TERMINATED BY '&\n';
-- File content: $t1t~tat&, where LINES STARTING BY($),FIELDS TERMINATED BY(~),ENCLOSED
BY(t),LINES TERMINATED BY(&\n)
-- Export data to the dumpfile file.
user_var=# SELECT * FROM t INTO DUMPFILE '/home/gaussdb/t.txt';
File content: 1a
-- Switch to the initial database. Replace init_db with the actual initial database name.
user_var=# \c init_db;
-- Delete the database.
gaussdb=# DROP DATABASE user_var;
```

- **FROM clause**

  Specifies one or more source tables for SELECT.

  The **FROM** clause can contain the following elements:

  – table_name

  Specifies the name of a table or view. The schema name can be added before the table name or view name, for example, schema_name.table_name.

NOTE

You can use DATABASE LINK to perform operations on remote tables and synonyms. For details, see **DATABASE LINK**.

- alias

  Gives a temporary alias to a table to facilitate the quotation by other queries.

  An alias is used for brevity or to eliminate ambiguity for self-joins. If an alias is provided, it completely hides the actual name of the table.

  NOTICE

  If an alias is specified for the *joined_table* table created by JOIN and *joined_table* is wrapped with "()", that is, (*joined_table*), non-reserved keywords UNPIVOT and PIVOT cannot be used as aliases.

- TABLESAMPLE sampling_method (argument [, ...]) [REPEATABLE (seed)]

  The TABLESAMPLE clause following **table_name** specifies that the specified **sampling_method** should be used to retrieve the subset of rows in the table.

  The optional REPEATABLE clause specifies the number of seeds used to generate random numbers in the sampling method. The seed value can be any non-null constant value. If the table was not changed during the query, the two queries having the same seed and **argument** values will select the same sampling in this table. However, different seed values usually generate different samples. If **REPEATABLE** is not specified, a new random sample will be selected for each query based on the seed generated by the system.

- TIMECAPSULE { TIMESTAMP | CSN } expression

  Queries the table data of a specified CSN or at a specified time point.

  Currently, the following tables do not support flashback query: system catalogs, DFS tables, global temporary tables, local temporary tables, unlogged tables, views, sequence tables, hash bucket tables, shared tables, and inherited tables.

  - TIMECAPSULE TIMESTAMP

    Searches for the result set of a specified time point based on the date as the flashback query flag. *date* must be a valid past timestamp.

  - TIMECAPSULE CSN

    Searches for the result set of a specified CSN based on the CSN flashback of the table as the flashback query flag. The CSN can be obtained from **snpcsn** recorded in **gs_txn_snapshot**.

  - expression

    Constants, functions, or SQL expressions.

NOTE

- A flashback query cannot span statements that affect the table structure or physical storage. Otherwise, an error is reported. Between the flashback point and the current point, if a statement (TRUNCATE, DDL, DCL, or VACUUM FULL) has been executed to modify the table structure or affect physical storage, the flashback fails.
- Flashback query supports scanning PCR UB-tree indexes. If no such indexes, flashback query supports only seqScan for full table scanning.
- When the flashback point is too old, the old version cannot be obtained because the flashback version is recycled. As a result, the flashback fails and the error message "Restore point too old" is displayed.
- The flashback point is specified by time. The maximum difference between the flashback point and the actual time is 3 seconds.
- After truncating a table, perform a flashback query or flashback on the table. The error message "Snapshot too old" is displayed when a flashback is performed at a specified time point. Data cannot be found or the error message "Snapshot too old" is reported during the CSN-based flashback.

– column_alias

  Specifies the column alias.

– PARTITION

  Queries data in the specified partition in a partitioned table.

– partition_name

  Specifies the name of a partition.

– partition_value

  Specifies the value of the specified partition key. If there are many partition keys, use the **PARTITION FOR** clause to specify the value of the only partition key you want to use.

– SUBPARTITION

  Queries data in the specified level-2 partition in a partitioned table.

– subpartition_name

  Specifies the name of a level-2 partition name.

– subpartition_value

  Specifies the key values of specified level-1 and level-2 partitions. The values of the two partition keys specified by the **SUBPARTITION FOR** clause uniquely identify a level-2 partition.

– subquery

  Performs a subquery in the **FROM** clause. A temporary table is created to save subquery results.

– with_query_name

  Specifies that the **WITH** clause can also be used as the source of the **FROM** clause and can be referenced by the name of the **WITH** query.

– function_name

  Function name. Function calls can appear in the **FROM** clause.

– join_type

  The options are as follows:

- [INNER] JOIN

  A **JOIN** clause combines two **FROM** items. You can use parentheses to determine the order of nesting. In the absence of parentheses, **JOIN** nests left-to-right.

- LEFT [OUTER] JOIN

  Returns all rows that meet join conditions in the Cartesian product, plus those rows that do not match the right table rows in the left table by join conditions. This left-hand row is extended to the full width of the joined table by inserting **NULL** values for the right-hand columns. Note that only the **JOIN** clause's own condition is considered while the system decides which rows have matches. Outer conditions are applied afterward.

- RIGHT [OUTER] JOIN

  Returns all the joined rows, plus one row for each unmatched right-hand row (extended with **NULL** on the left).

  This is just a notational convenience, since you could convert it to a **LEFT OUTER JOIN** by switching the left and right inputs.

- FULL [OUTER] JOIN

  Returns all the joined rows, pluses one row for each unmatched left-hand row (extended with **NULL** on the right), and pluses one row for each unmatched right-hand row (extended with **NULL** on the left).

- CROSS JOIN

  Is equivalent to **INNER JOIN ON (TRUE)**, which means no rows are removed by qualification. These join types are just a notational convenience, since they do nothing you could not do with plain **FROM** and **WHERE**.

  NOTE

  For the **INNER** and **OUTER** join types, a join condition must be specified, namely exactly one of **NATURAL ON**, **join_condition**, or **USING (join_column [, ...])**. For **CROSS JOIN**, none of these clauses can appear.

  You can use JOIN consecutively.

CROSS JOIN and **INNER JOIN** produce a simple Cartesian product, the same result as you get from listing the two items at the top level of **FROM**.

– ON join_condition

  Defines which rows have matches in joins. Example: ON left_table.a = right_table.a You are advised not to use numeric types such as int for **join_condition**, because such types can be implicitly converted to bool values (non-zero values are implicitly converted to **true** and **0** is implicitly converted to **false**), which may cause unexpected results.

– USING(join_column[, ...])

  ON left_table.a = right_table.a AND left_table.b = right_table.b ... abbreviation. Corresponding columns must have the same name.

– NATURAL

Is a shorthand for a **USING** list that mentions all columns in the two tables that have the same names.

– from item

Specifies the name of the query source object connected.

– xmltable_clause

Currently, xmltable_clause can be used only in A or PG compatibility mode. xmltable_clause produces a virtual table based on data of the given XMLTYPE (in A compatibility mode) or XML (in PG compatibility mode) type. The syntax format is as follows:

```
XMLTABLE(
xmlnamespaces_clause
row_expression
passing_clause
columns_clause
)
```

- For xmlnamespaces_clause:

```
[
   XMLNAMESPACES(
   {string AS identifier }
   |
   { DEFAULT string }
   [, { string AS identifier } | { DEFAULT string } ]...
   ),
]
```

The optional xmlnamespaces_clause is a clause that starts with XMLNAMESPACES followed by a comma-separated XML namespace declaration. **string** indicates the full name of the namespace for which single quotation marks are used as boundary characters. **identifier** indicates the namespace alias for which double quotation marks are used as boundary characters. The alias can be used in row_expression and columns_clause. In the current version, DEFAULT cannot be used to declare the default namespace.

- row_expression

The required **row_expression** is an XPath 1.0 expression with English single quotation marks as boundary characters. The document_expression in the subsequent passing_clause is used as its context to obtain a group of XML nodes. These nodes are processed by columns_clause to generate each row of the virtual table.

- For passing_clause:

```
PASSING [BY { REF | VALUE }] document_expression [BY { REF | VALUE }]
```

The required **passing_clause** starts with PASSING, where **document_expression** is the data of the XMLTYPE (in A compatibility mode) or XML (in PG compatibility mode) type that is processed in the transferred **XMLTABLE**. Currently, only single-root data is supported. In A compatibility mode, an error is reported regardless of whether BY VALUE or BY REF in the first or second place is used. In PG compatibility mode, although BY VALUE or BY REF in the first and second places can be identified and received, no functional processing is performed.

- For columns_clause:

```
[
   COLUMNS
```



```
    name { type [PATH column_expression] [DEFAULT default_expression] [NOT NULL |
NULL] | FOR ORDINALITY }
    [, ...]
]
```

The optional **columns_clause** is used to specify the information required for the columns to be generated in the virtual table.

- **name** indicates the name of a column.
- **type** indicates the column type.
- **PATH**: **column_expression** in the optional **PATH** part is an expression of XPath 1.0. A node in the node set obtained by **row_expression** is used as the context of the node, and the data required by the value of the corresponding column in the row data generated by the node is obtained through processing by the node. There is an implicit conversion when it is converted to a result of the TYPE type. If the **PATH** part is not provided, **name** is regarded as **column_expression**.
- **DEFAULT**: **default_expression** in the optional **DEFAULT** part is an expression. If a NULL value is obtained after **column_expression** processing, the result obtained by calculating **default_expression** is used to generate the value of the column. Note that **default_expression** is calculated only when necessary. If the expression is stable, **default_expression** is calculated only once when necessary.
- **NOT NULL** | **NULL**: In PG compatibility mode, the optional NOT NULL or NULL is used to indicate whether the column value can be **NULL**. If the column value is inconsistent with the specified option, an error is reported.
- **FOR ORDINALITY**: indicates that the column is a row number column. Row numbers starting from 1 are filled for generated rows. Only one column can be marked as **FOR ORDINALITY**.
- In A compatibility mode, **columns_clause** can be omitted. After **columns_clause** is omitted, the default **columns_clause: COLUMNS column_value XMLTYPE PATH '.'** is generated internally for subsequent processing. In PG compatibility mode, **columns_clause** cannot be omitted.

NOTICE

Currently, xmltable_clause has the following restrictions:

- The involved XPath expressions are XPath 1.0.
- Currently, XPath expressions such as '**..**' cannot be used in **column_expression**.
- When the data obtained through **column_expression** is converted to the data of the TYPE type, in A compatibility mode, if the obtained data is too long and exceeds the typmod of the type, the data will be truncated. Currently, long data is truncated only when the typmod of the CHAR, CHARACTER, NCHAR, BPCHAR, VARCHAR, CHARACTER VARYING, VARCHAR2 and NVARCHAR2 data types (including those attached with (n)) is greater than 0. In PG compatibility mode, an error is reported.
- Currently, the maximum size of the input XMLTYPE (in A compatibility mode) or XML (in PG compatibility mode) type data is 1 GB.
- For upgrade from a version that does not support XMLTABLE to a version that supports XMLTABLE, the XMLTABLE syntax cannot be used during the upgrade observation period.
- The following expression is supported in XPath only in A compatibility mode: *:nodename, where **nodename** indicates the node name. This expression indicates that the namespace of the node specified by **nodename** is ignored when the node is selected.
- The non-reserved keyword XMLTABLE cannot be used as the name of a function of the Functions as Table Sources type.

NOTE

When non-constructed XMLTYPE (in A compatibility mode) data is entered, spaces and carriage return characters that are at the same level as the non-constructed part and are used to control the writing format between nodes are parsed as text nodes. For example, in the following text, the space and carriage return between the end label of **node2** and the start label of **node3** are parsed as text nodes, which is different from that in database A.

Input:

```
<root>
<node1>node1</node1>
malform
<node2>node2</node2>
<node3>node3</node3>
</root>
```

Actual situation:

```
<root><node1>node1</node1>
malform
<node2>node2</node2>
<node3>node3</node3>
</root>
```


Currently, the functions implemented by xmltable_clause are different from those implemented by databases A and PG. Pay attention to the differences. For details, see **Table 7-226**.

Table 7-226 Comparison between GaussDB Kernel and databases A and PG

GaussDB Kernel	PostgreSQL	Database A
XPath 1.0 expressions are used at **row_expression** and **PATH** in **columns_clause**.	Same as that in GaussDB Kernel.	XQuery 1.0 expressions are used at **row_expression** and **PATH** in **columns_clause**.
The default namespace function is not supported.	Same as that in GaussDB Kernel.	The default namespace function is supported.
A single data record can be transferred through passing_clause, but the alias cannot be used.	Same as that in GaussDB Kernel. (Note that the input data is of the XML type.)	Multiple data records can be transferred through the corresponding clause and aliases can be used.
The passing_clause cannot be omitted.	Same as that in GaussDB Kernel.	The corresponding clause can be omitted.
The RETURNING SEQUENCE BY REF clause cannot be used after passing_clause.	Same as that in GaussDB Kernel.	The RETURNING SEQUENCE BY REF clause can be used after the corresponding clause.
In **columns_clause**, (SEQUENCE) BY REF cannot be used to modify the returned data of the XMLType type.	Same as that in GaussDB Kernel.	You can use (SEQUENCE) BY REF in the corresponding clause to modify the returned data of the XMLType type.

GaussDB Kernel	PostgreSQL	Database A
If the **PATH** part in **columns_clause** is omitted and the column name is not enclosed using double quotation marks (""), the node can be correctly found for subsequent operations when the node name in the data transferred to XMLTABLE is in lowercase.	Same as that in GaussDB Kernel.	If the corresponding part is omitted and the column name is not enclosed using double quotation marks (""), the node can be correctly found for subsequent operations when the node name in the data transferred to XMLTABLE is in uppercase.
The declaration of the column type in **columns_clause** cannot be omitted.	Same as that in GaussDB Kernel.	The declaration of the column type in the corresponding clause can be omitted.
In A compatibility mode, the **columns_clause** can be omitted. In PG compatibility mode, the **columns_clause** cannot be omitted.	The corresponding clause cannot be omitted.	The corresponding clause can be omitted.
In A compatibility mode, if the returned data length exceeds the typmod of the type, the data is truncated. In PG compatibility mode, if the returned data length exceeds the typmod of the type, an error is reported.	An error is reported when the length of the returned data exceeds the typmod of the type.	If the length of the returned data exceeds the value of typmod, the data is truncated.

Example:

```
gaussdb=# SELECT * FROM XMLTABLE(
gaussdb(# XMLNAMESPACES('nspace1' AS "ns1", 'nspace2' AS "ns2"), -- Declare two XML namespaces 'nspace1' and 'nspace2' and the corresponding aliases "ns1" and "ns2".
gaussdb(# '/ns1:root/*:child' -- Select the root node whose namespace is 'nspace1' from the transferred data through row_expression, select all child nodes under the root node, and ignore the child namespaces. ns1 is the alias of 'nspace1'.
gaussdb(# PASSING xmltype(
gaussdb(# '<root xmlns="nspace1">
gaussdb'#     <child>
gaussdb'#         <name>peter</name>
gaussdb'#         <age>11</age>
gaussdb'#     </child>
```



```
gaussdb'#     <child xmlns="nspace1">
gaussdb'#         <name>qiqi</name>
gaussdb'#         <age>12</age>
gaussdb'#     </child>
gaussdb'#     <child xmlns="nspace2">
gaussdb'#         <name>hacker</name>
gaussdb'#         <age>15</age>
gaussdb'#     </child>
gaussdb'# </root>')
gaussdb(# COLUMNS
gaussdb(# columns FOR ORDINALITY, -- Row number column
gaussdb(# name varchar(10) path 'ns1:name', -- Select the name node whose namespace is 'nspace1' from each child node obtained by row_expression, convert the value in the node to varchar(10), and return the value. ns1 is the alias of nspace1.
gaussdb(# age int); -- Select the age node from each child node obtained by row_expression, convert the value in the node to int, and return the value. The first child node does not explicitly specify the namespace. Therefore, the namespace of the parent node root is followed. Therefore, no value in this column is returned.
 column | name  | age
--------+-------+-----
      1 | peter |
      2 | qiqi  |
      3 |       |
(3 rows)
```

– unpivot_clause

  Converts a column to a row. The syntax format is as follows:

```
UNPIVOT [ {INCLUDE | EXCLUDE} NULLS ]
  (
     unpivot_col_clause
     unpivot_for_clause
     unpivot_in_clause
  )
```

  - {INCLUDE | EXCLUDE} NULLS

    Controls whether the converted result contains NULL rows. **INCLUDE NULLS** indicates that the converted results contain NULL rows. **EXCLUDE NULLS** indicates that the converted results do not contain NULL rows. If this clause is ignored, the unpivot operation removes NULL rows from the converted results by default.

  - unpivot_col_clause

```
unpivot_col_element
```

    unpivot_col_element specifies the output column names. These columns store the values to be converted.

  - unpivot_col_element

```
{ column | ( column [, column]... ) }
```

    unpivot_col_element has two forms: column and (column [, column]...).

  - unpivot_for_clause

```
FOR { unpivot_col_element }
```

    unpivot_col_element contained in the **unpivot_for_clause** is used to specify the output column names. These columns store the alias and names of the column to be converted.

  - unpivot_in_clause

```
IN ( unpivot_in_element [,unpivot_in_element...] )
```

    unpivot_in_clause specifies the columns to be converted. The column names and column values are saved in the previously specified output columns.

unpivot_in_element

```
{ unpivot_col_element }[  AS { unpivot_alias_element } ]
```

unpivot_col_element specifies the column to be converted. If (column [, column]...) is used to specify the column to be converted, the *column* names are connected by underscores (_) and saved in the output columns. For example, IN ((col1, col2)) generates the column name **col1_col2** and saves it in the output column specified by **unpivot_for_clause**. In addition, the AS keyword can be used to specify an alias for the column to be converted. Once an alias is specified, the alias is saved in the output column instead of the name of column to be converted.

- unpivot_alias_element

  ```
  { alias | ( alias [, alias]... )}
  ```

  Similar to **unpivot_col_element**, **unpivot_alias_element** has two forms. **alias** indicates the specified alias.

NOTICE

Currently, **unpivot_clause** has the following restrictions:

- This parameter can be used only in A compatibility mode.
- The **unpivot_clause** clause cannot be used with hints.
- For **unpivot_col_clause**, the number of output columns specified by **unpivot_col_element** must be the same as that of **unpivot_col_element** contained in **unpivot_in_clause**.
- For **unpivot_for_clause**, the number of output columns specified by **unpivot_col_element** must be the same as the number of aliases specified by **unpivot_alias_element** contained in **unpivot_in_clause**.
- For **unpivot_in_clause**, the alias must be a constant or an expression that can be converted to a constant.
- For **unpivot_in_clause**, constant expressions support only IMMUTABLE functions.
- For all **unpivot_col_element** parameters contained in **unpivot_in_clause**, if the column types in the same position of these **unpivot_col_element** parameters are different, UNPIVOT attempts to convert the column types in order to convert the values of columns to be converted to a public type. Similarly, for all **unpivot_alias_element** parameters, if the alias types of these **unpivot_alias_element** parameters in the same position are different, UNPIVOT attempts to do the similar type conversion.

  For example, assume that there is an **unpivot_in_clause** in the form of IN (col1, col2), where col1 is of the int type and col2 is of the float type, UNPIVOT attempts to convert the column value of col1 to the public type float during the calculation.

– pivot_clause

Converts a row into column. The syntax format is as follows:

```
PIVOT [ XML ]
( aggregate_function ( expr ) [[AS] alias ]
   [, aggregate_function ( expr ) [[AS] alias ] ]...
```



```
  pivot_for_clause
  pivot_in_clause
)
```

- aggregate_function (expr) [[AS] alias]

  Aggregates calculation on a given expression. The calculation result is saved in the output column specified by **pivot_in_clause**. [AS] alias (The AS keyword can be omitted.) can be used to specify an alias for **aggregate_function**. The alias is added to the end of output column name specified by **pivot_in_clause** in the format of "*_alias*".

- pivot_for_clause

  ```
  FOR { column
    | ( column [, column]... )
    }
  ```

  Specifies the row to be converted. The **column** indicates a column of the row to be converted.

- pivot_in_clause

  ```
  IN ( { { { expr
         | ( expr [, expr]... )
         } [ [AS] alias]
       }...
     }
  )
  ```

  Specifies the name of the output column. The column name can consist of one or more expressions, for example, (expr1, expr2). When a column name consists of multiple expressions, these expressions are connected by underscores (_) in sequence. That is, the output column name corresponding to (expr1, expr2) is "expr1_expr2". These expressions not only generate output column names, but also determine the time when the aggregate function is triggered. If the values of row to be converted is the same as the value of these expressions, the results calculated by **aggregate_function** are saved in the output column names that consist of these expressions. Assume that expr1 is "1" and expr2 is "2". For row "1 2", **aggregate_function** is used for calculation. For row "1 1", the calculation is not triggered.

NOTICE

Currently, **pivot_clause** has the following restrictions:

- This parameter can be used only in A compatibility mode.
- The **pivot_clause** clause cannot be used with hints.
- If more than one **aggregate_function** is specified, at most one **aggregate_function** is allowed to have no alias, the rest of the **aggregate_function** functions are required to specify an alias.
- XML supports only syntax but does not support functions.
- The expression in **pivot_in_clause** can be a constant or an expression that can be converted to a constant. If the expression is not a unary expression, specify an alias for the expression.
- For **pivot_in_clause**, constant expressions support only IMMUTABLE functions.
- For the expression in **pivot_in_clause**, when the keyword AS is used to specify an alias for the expression, only the non-reserved keywords can be used as aliases.
- If the length of an output column name exceeds 63 characters, an error is reported.

- **WHERE clause**

  Forms an expression for row selection to narrow down the query range of **SELECT**. **condition** indicates any expression that returns a value of Boolean type. Rows that do not meet this condition will not be retrieved. You are advised not to use numeric types such as int for **condition**, because such types can be implicitly converted to bool values (non-zero values are implicitly converted to **true** and **0** is implicitly converted to **false**), which may cause unexpected results.

  In the **WHERE** clause, you can use the operator (+) to convert a table join to an outer join. However, this method is not recommended because it is not the standard SQL syntax and may raise syntax compatibility issues during platform migration. There are many restrictions on using the operator (+):

  a. It can appear only in the **WHERE** clause.

  b. If a table join has been specified in the **FROM** clause, the operator (+) cannot be used in the **WHERE** clause.

  c. The operator (+) can work only on columns of tables or views, instead of on expressions.

  d. If table A and table B have multiple join conditions, the operator (+) must be specified in all the conditions. Otherwise, the operator (+) will not take effect, and the table join will be converted into an inner join without any prompt information.

  e. Tables specified in a join condition where the operator (+) works cannot cross queries or subqueries. If tables where the operator (+) works are not in the **FROM** clause of the current query or subquery, an error will be reported. If a peer table for the operator (+) does not exist, no error will be reported and the table join will be converted into an inner join.

  f. Expressions where the operator (+) is used cannot be directly connected through **OR**.

g. If a column where the operator (+) works is compared with a constant, the expression becomes a part of the JOIN condition.

h. A table cannot have multiple foreign tables.

i. The operator (+) can appear only in the following expressions: comparison, NOT, ANY, ALL, IN, NULLIF, IS DISTINCT FROM, and IS OF. It is not allowed in other types of expressions. In addition, these expressions cannot be connected through **AND** or **OR**.

j. The operator (+) can be used to convert a table join only to a left or right outer join, instead of a full join. That is, the operator (+) cannot be specified on both tables of an expression.

NOTICE

- For the WHERE clause, if special character "%", "_", or "\" is queried in LIKE, add the slash "\" before each character.
- For a hierarchical query, the WHERE expression is processed as follows in a multi-table join query:

  Decompose the WHERE expression based on the disjunctive and conjunctive actions and check whether each subexpression involves multiple tables (potential join conditions) queried at the current layer. If a subexpression is not a sublink and involves only multiple tables queried at the current layer, push it down to the non-recursive START WITH clause or the recursive CONNECT BY clause of the hierarchical query for prior execution as the JOIN condition. Otherwise, place it after the recursive CONNECT BY clause for execution as a filter condition.

  During internal implementation, the subexpressions that do not meet pushdown requirements in the parse tree of the original WHERE expression are deleted to obtain the final expression that can be pushed down. Then, the subexpressions that meet the pushdown condition are removed from the parse tree of the original WHERE expression to obtain the final expression that is not pushed down.

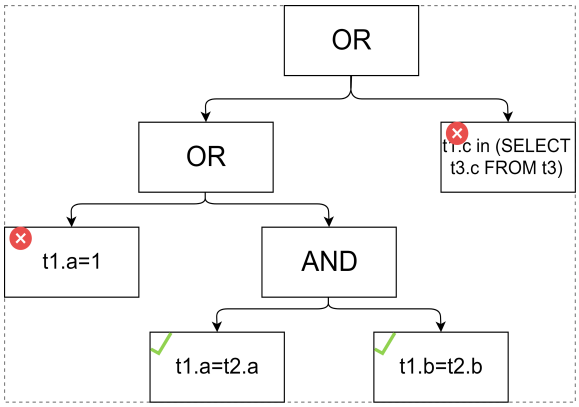
  To sum up, pay attention to the following points:

  - When the WHERE expression is split, the disjunctive action is also split. Therefore, the two expressions joined by OR are also split.
  - Sublinks must not be pushed down.
  - If an input column is not queried at the current layer, the column will not be pushed down.

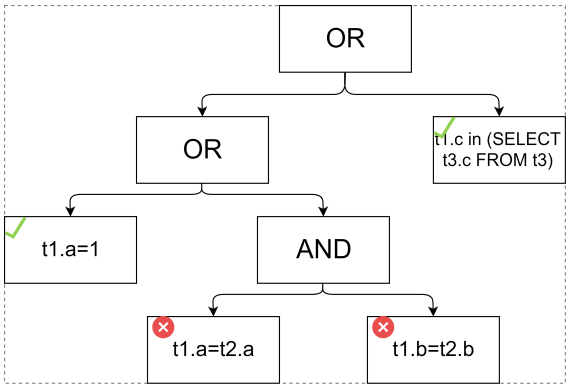
  Then, for the following example:

```
SELECT * FROM t1,t2
WHERE t1.a=1 or t1.a=t2.a and t1.b=t2.b or t1.c in (SELECT t3.c FROM t3)
START WITH t1.c=1
CONNECT BY PRIOR t2.c=t1.c;
```

  The following figure shows the pushdown conditions after the WHERE expression is split disjunctively and conjunctively.



  It is equivalent to "t1.a=t2.a AND t1.b=t2.b". The following figure shows the conditions that are not pushed down.



It is equivalent to "t1.a=1 OR t1.c in (SELECT t3.c FROM t3)".

For another example:

```
SELECT * FROM t3 WHERE EXISTS(
SELECT * FROM t1,t2
WHERE t1.a+t2.a=t3.a
START WITH t1.c=1
CONNECT BY PRIOR t2.c=t1.c;
)
```

The WHERE condition contains **t3.a** inputted by the outer query. Therefore, the WHERE condition is not pushed down.

- **START WITH clause**

  The **START WITH** clause is usually used together with the **CONNECT BY** clause and indicates the initial condition of recursion. Data is traversed recursively and hierarchically. If this clause is omitted and the **CONNECT BY** clause is used alone, all rows in the table are used as the initial set. For details, see **•CONNECT BY**.

- **CONNECT BY clause**

  CONNECT BY indicates a recursive join condition. It is used together with START WITH to implement data traversal and recursion. Example:

```
gaussdb=#  CREATE TABLE test(name varchar, id int, fatherid int);
gaussdb=#  INSERT INTO test VALUES('A', 1, 0), ('B', 2, 1),('C',3,1),('D',4,1),('E',5,2);
gaussdb=#  SELECT * FROM TEST START WITH id = 1 CONNECT BY prior id = fatherid ORDER
SIBLINGS BY id DESC;
 name | id | fatherid
------+----+----------
 A    | 1 |        0
 D    | 4 |        1
 C    | 3 |        1
 B    | 2 |        1
 E    | 5 |        2
(5 rows)
```

  In the CONNECT BY condition, the PRIOR keyword can be specified for a column to indicate that the column is recursive. If NOCYCLE is added before the recursive join condition, recursion stops when a loop record is encountered. (Note: A SELECT statement containing the START WITH .. CONNECT BY clause does not support the FOR SHARE or UPDATE lock.)

  The process of executing the START WITH statement is as follows:

  a. The initial dataset is selected based on the condition in the START WITH clause. In the preceding example, **('A', 1, 0)** is selected first. Then, this initial dataset is set as the working set.

  b. If the working set is not empty, the data in the working set is used as the input for the next query. The filter criteria are specified in the CONNECT

BY clause. The keyword PRIOR indicates the current record. For example, **prior id = fatherid** in the preceding example indicates that the ID of the current record is the **fatherid** of the next record.

c. Set the dataset filtered in step 2 as the working set and repeat the operation in step 2.

In addition, the database adds the following pseudocolumns to each selected data record so that users can learn about the location of the data in the recursive or tree structure:

- **LEVEL**: node level.
- **CONNECT_BY_ISLEAF**: specifies whether a node is a leaf node.

NOTICE

connect_by_isleaf: If a node does not have any subnode, the node is a leaf node and **connect_by_isleaf** is set to **1**. Otherwise, **connect_by_isleaf** is set to **0**.

Assume that table T1 exists and the data in table T1 is shown in **Figure 7-12**.

Figure 7-12 Data structure

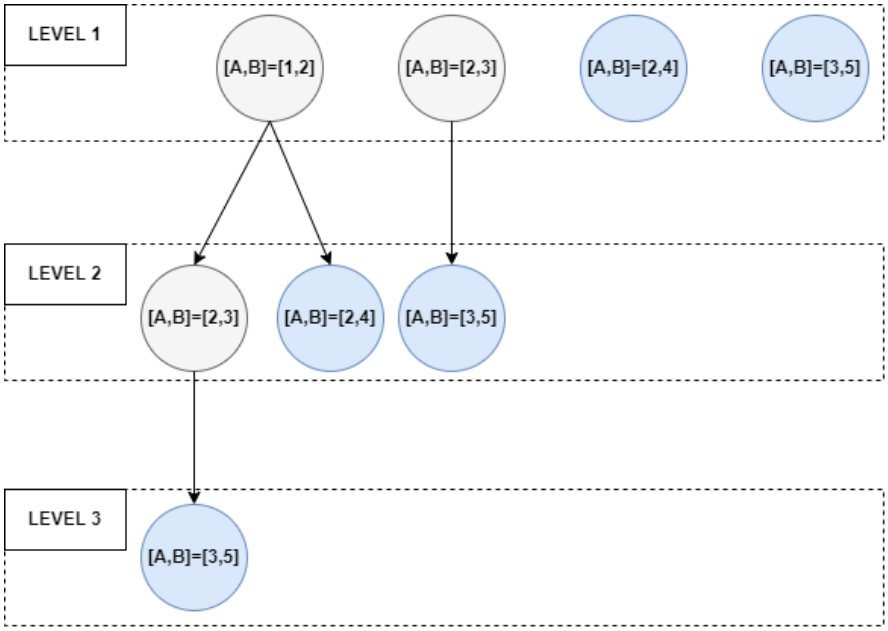
T1	
A	B
1	2
2	3
2	4
3	5

Run the following statement:

```
SELECT * FROM T1 CONNECT BY PRIOR B=A AND LEVEL<=3;
```

The tree structure of the result is shown in **Figure 7-13**, where the blue nodes are leaf nodes.

Figure 7-13 Logical structure of the execution



In addition to pseudocolumns, the following query functions are provided (for details, see **Hierarchical Recursion Query Functions**):

- **sys_connect_by_path(col, separator)**: returns the connection path from the root node to the current row. The **col** parameter indicates the name of the column displayed in the path, and the **separator** parameter indicates the connector.
- **connect_by_root(col)**: displays the root node of the node. **col** indicates the name of the output column.

If a loop exists in the dataset, the database provides loop detection. By default, if a loop is detected, an error is reported and no data is returned. In addition, the NOCYCLE keyword is provided. With it, the query can be executed normally and when the first duplicate data record is found, the query exits directly instead of reporting an error.

Besides, in the hierarchical query, the search is performed strictly according to the depth-first order. If ROWNUM is used as the filtering condition in START WITH or CONNECT, the value of **ROWNUM** is increased by 1 for each record to be returned. Then, the record is verified based on **ROWNUM** conditions. Records that do not satisfy the conditions are discarded and the value of **ROWNUM** is decreased by 1.

NOTICE

- The PRIOR keyword can be used only in the CONNECT BY clause instead of the START WITH clause.
- In addition to **targetlist**, "prior (single column)" is parsed as "prior single column". User-defined functions named **prior** are not perceived.
- The PRIOR keyword can be specified only for columns in the table instead of expressions, pseudocolumns, or type conversion. For example, PRIOR (a + 1) is not allowed.
- In the CONNECT BY clause, the column using the keyword PRIOR cannot be in the same condition with pseudocolumns such as level and rownum, but they can be in different conditions. For example, (PRIOR a = level) is not allowed, but (PRIOR a = b) and (level = 1) is allowed. Different conditions refer to the conditions connected by AND at the top of the CONNECT BY clause. For example, (PRIOR a = 1 or level = 1) is considered as a condition and is not allowed.
- In the START WITH and CONNECT BY clauses, pseudocolumns cannot be used for sublinks, for example, "rownum = (subquery)" or "rownum in (subquery)".
- You are advised not to use pseudo columns in the CONNECT BY statement. If pseudocolumns need to be used, you need to test the columns to avoid inconsistency between the result and expected result.
- When START WITH and CONNECT BY are called on the CTE defined by WITH AS, if there are multiple CTEs, ensure that the definition of each CTE does not depend on other CTEs.
- If no loop exists in the data but the error message "runs into cycle" is reported, increase the value of **max_recursive_times**.
- **connect_by_isleaf**, **connect_by_iscycle** and **level** are of the int type.
- In hierarchical queries, **connect_by_isleaf**, **connect_by_iscycle**, and **level** must be parsed as pseudocolumns and will not be parsed as columns whose alias names in projection columns are the same as pseudocolumns.
- You are advised not to reference other views when defining views with hierarchical queries.
- Optimization suggestions for the START WITH clause:
  - Create indexes based on the conditions in the CONNECT BY clause to improve the performance of the START WITH clause.
  - Identify bottlenecks based on the plan collected by running EXPLAIN PERFORMANCE or in WDR. If the recursive operator (inner plan) of RECURSIVE UNION is the Hash JOIN operator, but the hash table is created for the temporary table **tmp_result** or the hash table in plan is materialized (that is, the batch size is greater than 1), the possible cause is that the value of **work_mem** is too small. As a result, the hash table cannot be created for the outer data table. You can increase the value of **work_mem** to improve performance.

    Note: GaussDB optimizes tables with a small volume of data and caches table results in hash tables to improve performance. In this case, indexes are not required. However, if the data volume exceeds

the limit specified by **work_mem**, the optimization becomes invalid. In this case, you can create indexes for optimization.

- **ORDER SIBLINGS BY clause**

  The output of the START WITH statement is returned level by level. However, there is no sequence guarantee at each level because the database automatically selects the optimal execution path during each round of query. In the preceding example, A is output first, but the sequence of B, C, and D is not fixed. If you have requirements on the final output sequence, you can use ORDER SIBLINGS BY. The usage of ORDER SIBLINGS BY is the same as that of ORDER BY. ORDER SIBLINGS BY is used for sorting at each level during recursion.

  NOTICE

  The expression following ORDER SIBLINGS BY only supports sorting by calling non-aggregate and window functions for common columns, column name offsets, and column names. It does not support calling system functions related to START WITH for column names or using pseudocolumns related to START WITH.

- **GROUP BY clause**

  Condenses query results into a single row all selected rows that share the same values for the grouped expressions.

  – CUBE ({ expression | (expression [, ...]) } [, ...])

    A CUBE grouping is an extension to the **GROUP BY** clause that creates subtotals for all of the possible combinations of the given list of grouping columns (or expressions). In terms of multidimensional analysis, CUBE generates all the subtotals that could be calculated for a data cube with the specified dimensions. For example, given three expressions (n=3) in the CUBE clause, the operation results in $2^n = 2^3 = 8$ groupings. Rows grouped on the values of *n* expressions are called regular rows, and the rest are called superaggregate rows.

  – GROUPING SETS (grouping_element [, ...])

    Another extension to the **GROUP BY** clause. It allows users to specify multiple **GROUP BY** clauses. This improves efficiency by trimming away unnecessary data. After you specify the required data group, the database does not need to compute a whole **ROLLUP** or **CUBE**.

  NOTICE

  - If a SELECT list expression quotes some ungrouped fields and no aggregate function is used, an error is displayed. This is because multiple values may be returned for ungrouped fields.
  - If a SELECT list expression references a constant, the GROUP BY clause does not need to group the constant. Otherwise, an error is reported.

- **HAVING clause**

Selects special groups by working with the **GROUP BY** clause. The **HAVING** clause compares some attributes of groups with a constant. Only groups that matching the logical expression in the **HAVING** clause are extracted.

- **WINDOW clause**

  The general format is **WINDOW window_name AS (window_definition) [, ...]**. **window_name** is a name can be referenced by **window_definition**. **window_definition** can be expressed in the following forms:

  ```
  [ existing_window_name ]
  [ PARTITION BY expression [, ...] ]
  [ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST | LAST } ] [, ...] ]
  [ frame_clause ]
  ```

  frame_clause defines a **window frame** for the window function. The window function (not all window functions) depends on **window frame** and **window frame** is a set of relevant rows of the current query row. **frame_clause** can be expressed in the following forms:

  ```
  [ RANGE | ROWS ] frame_start
  [ RANGE | ROWS ] BETWEEN frame_start AND frame_end
  ```

  frame_start and **frame_end** can be expressed in the following forms:

  ```
  UNBOUNDED PRECEDING
  VALUE PRECEDING
  CURRENT ROW
  VALUE FOLLOWING
  UNBOUNDED FOLLOWING
  ```

- **UNION clause**

  Computes the set union of the rows returned by the involved **SELECT** statements.

  The **UNION** clause has the following constraints:

  – By default, the result of **UNION** does not contain any duplicate rows unless the **ALL** clause is declared.

  – Multiple **UNION** operators in the same SELECT statement are evaluated left to right, unless otherwise specified by parentheses.

  – **FOR UPDATE**, **FOR NO KEY UPDATE**, **FOR SHARE**, and **FOR KEY SHARE** cannot be specified in the result or input of **UNION**.

  – The GUC parameter **enable_union_all_order** determines whether to preserve the order of UNION ALL subqueries.

    - Enabling: **set enable_union_all_order to on**
    - Disabling: **set enable_union_all_order to off**
    - Querying: **show enable_union_all_order**

    When the **enable_union_all_order** parameter is enabled, subquery order preserving is supported if **UNION** is set to **ALL** and the main query is not sorted. In other cases, subquery order preserving is not supported.

  General expression:

  ```
  select_statement UNION [ALL] select_statement
  ```

  – **select_statement** can be any SELECT statement without the **ORDER BY**, **LIMIT**, **FOR UPDATE**, **FOR NO KEY UPDATE**, **FOR SHARE**, or **FOR KEY SHARE** clause.

  – **ORDER BY** and **LIMIT** can be attached to the subexpression if it is enclosed in parentheses.

- **INTERSECT clause**

  Computes the set intersection of rows returned by the involved **SELECT** statements. The result of **INTERSECT** does not contain any duplicate rows.

  The **INTERSECT** clause has the following constraints:

  – Multiple **INTERSECT** operators in the same **SELECT** statement are evaluated left to right, unless otherwise specified by parentheses.

  – Processing **INTERSECT** preferentially when **UNION** and **INTERSECT** operations are executed for results of multiple **SELECT** statements.

  General format:

  ```
  select_statement INTERSECT select_statement
  ```

  select_statement can be any SELECT statement without the **FOR UPDATE**, **FOR NO KEY UPDATE**, **FOR SHARE**, or **FOR KEY SHARE** clause.

- **EXCEPT clause**

  Has the following common form:

  ```
  select_statement EXCEPT [ ALL ] select_statement
  ```

  select_statement can be any SELECT statement without the **FOR UPDATE**, **FOR NO KEY UPDATE**, **FOR SHARE**, or **FOR KEY SHARE** clause.

  The **EXCEPT** operator computes the set of rows that are in the result of the left SELECT statement but not in the result of the right one.

  The result of **EXCEPT** does not contain any duplicate rows unless the **ALL** clause is declared. To execute **ALL**, a row that has *m* duplicates in the left table and *n* duplicates in the right table will appear MAX(*m*–*n*, 0) times in the result set.

  Multiple **EXCEPT** operators in the same **SELECT** statement are evaluated left to right, unless parentheses dictate otherwise. **EXCEPT** binds at the same level as **UNION**.

  Currently, the **FOR UPDATE**, **FOR NO KEY UPDATE**, **FOR SHARE**, and **FOR KEY SHARE** clauses cannot be specified for the result of **EXCEPT** or any input of **EXCEPT**.

- **MINUS clause**

  Has the same function and syntax as **EXCEPT** clause.

- **ORDER BY clause**

  Sorts data retrieved by **SELECT** in descending or ascending order. If the **ORDER BY** expression contains multiple columns:

  – If two columns are equal according to the leftmost expression, they are compared according to the next expression and so on.

  – If they are equal according to all specified expressions, they are returned in an implementation-dependent order.

  – When used with the **DISTINCT** keyword, the columns to be sorted in **ORDER BY** must be included in the columns of the result set retrieved by the SELECT statement.

  – When used with the **GROUP BY** clause, the columns to be sorted in **ORDER BY** must be included in the columns of the result set retrieved by the SELECT statement.

  When the clause is not used with the GROUP BY clause and the columns of the result set retrieved by the SELECT statement contain aggregate functions:

  - The ORDER BY clause is ignored if the columns of the retrieved result set do not contain any set returning functions.
  - Only the sorting of the columns that contain set returning functions is retained if the columns of the retrieved result set contain set returning functions.

  NOTE

  To support Chinese pinyin order, set the encoding format to **UTF-8**, **GB18030**, **GB18030_2022**, **GBK**, or **ZHS16GBK** when initializing the database. The statements are as follows:

  ```
  initdb –E UTF8 –D ../data –locale=zh_CN.UTF-8, initdb -E GB18030 -D ../data -locale=zh_CN.GB18030, initdb -E GB18030_2022 -D ../data -locale=zh_CN.GB18030, initdb –E GBK –D ../data –locale=zh_CN.GBK, or initdb –E ZHS16GBK –D ../data –locale=zh_CN.GBK.
  ```

- **LIMIT clause**

  Consists of two independent sub-clauses:

  LIMIT { count | ALL } limits the number of rows to be returned. **count** specifies the number of rows to be returned. The effect of **LIMIT ALL** is the same as that of omitting the LIMIT clause.

  OFFSET start count specifies the maximum number of rows to return, while **start** specifies the number of rows to skip before starting to return rows. When both are specified, **start** rows are skipped before starting to count the **count** rows to be returned.

  ROWNUM cannot be used as count or offset in the LIMIT clause.

- **OFFSET clause**

  The SQL: 2008 standard has introduced a different clause:

  OFFSET start { ROW | ROWS }

  start specifies the number of rows to skip before starting to return rows.

- **FETCH { FIRST | NEXT } [count] { ROW | ROWS } ONLY**

  If **count** is omitted in a **FETCH** clause, it defaults to **1**.

- **Locking clause**

  The **FOR UPDATE** clause locks the rows retrieved by **SELECT**. This prevents these rows from being modified or deleted by other transactions before the current transaction ends. That is, other transactions that attempt to run **UPDATE**, **DELETE**, **SELECT FOR UPDATE**, **SELECT FOR NO KEY UPDATE**, **SELECT FOR SHARE**, or **SELECT FOR KEY SHARE** for these rows will be blocked until the current transaction ends. Any **DELETE** on a row will also acquire the **FOR UPDATE** locking mode, as will **UPDATE** that modifies values on the primary key column. Conversely, **SELECT FOR UPDATE** waits for concurrent transactions that have run the preceding commands on the same row, and then locks and returns the updated row (there may be no row because the row may have been deleted).

  FOR NO KEY UPDATE behaves similarly to **FOR UPDATE**, except that it acquires a weaker lock that will not block **SELECT FOR KEY SHARE** that attempts to acquire the lock on the same row. Any **UPDATE** that does not acquire the **FOR UPDATE** lock will also acquire this locking mode.

  FOR SHARE behaves similarly, except that it acquires a shared rather than exclusive lock on each retrieved row. A shared lock blocks other transactions from executing **UPDATE**, **DELETE**, **SELECT FOR UPDATE**, or **SELECT FOR NO**

KEY UPDATE, but does not block **SELECT FOR SHARE** or **SELECT FOR KEY SHARE**.

FOR KEY SHARE is similar to **FOR SHARE** except that its lock is weak. **SELECT FOR UPDATE** is blocked but **SELECT FOR NO KEY UPDATE** is not blocked. A key-shared lock blocks other transactions from executing **DELETE** or **UPDATE** that modifies the key value, but does not block **UPDATE**, **SELECT FOR NO KEY UPDATE**, **SELECT FOR SHARE**, or **SELECT FOR KEY SHARE**.

To prevent the operation from waiting for the commit of other transactions, you can use **NOWAIT**. If the selected row cannot be locked immediately, an error is reported immediately and there is no waiting. If you use **WAIT N** and the selected row cannot be locked immediately, the operation needs to wait for *n* seconds (the value of *n* is of the int type with a range of $0 \leq n \leq 2147483$). If the lock is obtained within *n* seconds, the operation is performed normally. Otherwise, an error is reported. If you use **SKIP LOCKED**, locked rows are skipped when a table is locked. Only rows that are locked using a row lock can be skipped. In scenarios where locks do not block each other in different transactions, for example, SELECT FOR SHARE - SELECT FOR SHARE/FOR KEY SHARE, SELECT FOR KEY SHARE - SELECT FOR KEY SHARE/FOR NO KEY SHARE, rows that are locked with preceding locks are not skipped though **SKIP LOCKED** is specified.

If specified tables are named in a locking clause, then only rows coming from those tables are locked; any other tables used in **SELECT** are simply read as usual. Otherwise, locking all tables in the statement.

If a locking clause is applied to a view or sub-query, it affects all tables used in the view or sub-query.

Multiple locking clauses can be written if it is necessary to specify different locking behaviors for different tables.

If a table appears (or implicitly appears) in multiple clauses at the same time, the strongest lock is used. Similarly, a table is processed as **NOWAIT** if that is specified in any of the clauses affecting it.

NOTICE

- Only FOR SHARE and FOR UPDATE can be used to query the Ustore table.
- For the FOR UPDATE/SHARE statements whose subquery is a stream plan, the same locked row cannot be concurrently updated.
- The DATABASE LINK function does not support the SKIP LOCKED syntax.

- **NLS_SORT**

  Specifies that a field is sorted in a special order. Currently, only Chinese Pinyin and case-insensitive sorting are supported. To support this sorting mode, you need to set the encoding format to **UTF8**, **GB18030**, **GB18030_2022**, **GBK**, or **ZHS16GBK** when creating a database. If you set the encoding format to another format, for example, **SQL_ASCII**, an error may be reported or the sorting mode may be invalid.

  Value range:

  – **SCHINESE_PINYIN_M**, sorted by Pinyin order.

  – **generic_m_ci**: sorted in case-insensitive order (optional; only English characters are supported in the case-insensitive order.)

- **PARTITION clause**

  Queries data in the specified partition in a partitioned table.

7.12.18.3.1 Simple Queries

A simple query retrieves one or more columns of data from one or more tables or views.

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE student(
    sid INT PRIMARY KEY,
    class INT,
    name  VARCHAR(50),
    sex INT CHECK(sex = 0 OR sex = 1) -- Gender. 1: male; 0: female
);
gaussdb=# INSERT INTO student (sid, class, name, sex) VALUES (1, 1, 'Michael', 0);
gaussdb=# INSERT INTO student (sid, class, name, sex) VALUES (2, 2, 'Bob', 1);
gaussdb=# INSERT INTO student (sid, class, name, sex) VALUES (3, 2, 'Gary', 0);

-- Query some columns.
gaussdb=# SELECT sid, name FROM student;
 sid |  name
-----+---------
   1 | michael
   2 | bob
   3 | Gary
(3 rows)

-- Query all columns.
gaussdb=# SELECT * FROM student;
 sid | class |  name   | sex
-----+-------+---------+-----
   1 |     1 | michael |   0
   2 |     2 | bob     |   1
   3 |     2 | Gary    |   0
(3 rows)

-- Specify the column alias.
gaussdb=# SELECT sid student_id, name FROM student;
 student_id |  name
------------+---------
          1 | michael
          2 | bob
          3 | Gary
(3 rows)

-- Delete.
gaussdb=# DROP TABLE student;
```

7.12.18.3.2 Conditional Queries

A conditional query retrieves data that meets the conditions.

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE test_grade(
    id INT,
    name VARCHAR(20),
    score FLOAT       -- Score
);
gaussdb=# INSERT INTO test_grade VALUES (1,'Scott',90),(2,'Jack',87.5),(3,'Ben',48);

-- Query information about students who fail to pass the test.
gaussdb=# SELECT * FROM test_grade WHERE score < 60;
 id | name | score
----+------+-------
  3 | Ben  |    48
(1 row)
```



```
-- Delete.
gaussdb=# DROP TABLE test_grade;
```

7.12.18.3.3 Group Queries

A group query is usually used together with aggregate functions to query category statistics. Common aggregate functions include count(), sum(), avg(), min(), and max(). GROUP BY is often used in group queries.

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE student (id INT,name VARCHAR(20),class INT);
gaussdb=# INSERT INTO student VALUES (1, 'Scott', 1), (2, 'Ben', 1);
gaussdb=# INSERT INTO student VALUES (3, 'Jack', 2),(4, 'Anna', 2),(5, 'Judy', 2);
gaussdb=# INSERT INTO student VALUES (6, 'Sally', 3), (7, 'Jordan', 3);

-- Use the aggregate function COUNT() to count the number of people in each class.
gaussdb=# SELECT class, count(*) AS cnt FROM student GROUP BY class;
 class | cnt
-------+-----
     2 |   3
     1 |   2
     3 |   2
(3 rows)

-- Use the HAVING clause to retrieve data of classes with less than three people.
gaussdb=# SELECT class, COUNT(*) AS num FROM student GROUP BY class HAVING COUNT(*) < 3;
 class | num
-------+-----
     1 |   2
     3 |   2
(2 rows)

-- Delete.
gaussdb=# DROP TABLE student;
```

7.12.18.3.4 Pagination Queries

The syntax is as follows:

```
SELECT query_list FROM table_name [ LIMIT { [offset,] count | ALL } ]
```

- **offset**: indicates the number of rows to skip before starting to return rows.
- **count**: indicates the number of data records to be retrieved after the skipped rows.
- **ALL**: indicates that all data after the skipped rows is retrieved.

```
-- Create a table and insert 100 data records into the table.
gaussdb=# CREATE TABLE testl(id int PRIMARY KEY, flag varchar(10));
gaussdb=# INSERT INTO testl (id, flag) VALUES (generate_series(1,100),'flag'||generate_series(1,100));

-- Query the first five data records.
gaussdb=# SELECT * FROM testl ORDER BY 1  LIMIT 5;
 id | flag
----+-------
  1 | flag1
  2 | flag2
  3 | flag3
  4 | flag4
  5 | flag5
(5 rows)

-- Query four data records after the twentieth record.
gaussdb=# SELECT * FROM testl ORDER BY 1 LIMIT 20,4;
 id |  flag
----+--------
 21 | flag21
```



```
 22 | flag22
 23 | flag23
 24 | flag24
(4 rows)

-- Query all data after the ninety-sixth record.
gaussdb=# SELECT * FROM testl ORDER BY 1 LIMIT 96,ALL;
 id  |  flag
-----+---------
  97 | flag97
  98 | flag98
  99 | flag99
 100 | flag100
(4 rows)

-- Delete.
gaussdb=# DROP TABLE testl;
```

7.12.18.3.5 Partition Queries

Queries data in a specified partition.

```
-- Create a range partitioned table.
gaussdb=# CREATE TABLE test_range1(
    id INT,
    info VARCHAR(20)
) PARTITION BY RANGE (id) (
    PARTITION p1 VALUES LESS THAN (200),
    PARTITION p2 VALUES LESS THAN (400),
    PARTITION p3 VALUES LESS THAN (600),
    PARTITION p4 VALUES LESS THAN (800),
    PARTITION pmax VALUES LESS THAN (MAXVALUE)
);
-- Insert 1000 data records.
gaussdb=# INSERT INTO test_range1 VALUES(GENERATE_SERIES(1,1000),'abcd');

-- Query the number of data records in the p1 partition.
gaussdb=# SELECT COUNT(*) FROM test_range1 PARTITION (p1);
 count
-------
   199
(1 row)

-- Delete.
gaussdb=# DROP TABLE test_range1;
```

7.12.18.3.6 Join Queries

A join query may also be referred to as a cross-table query, where multiple tables need to be associated.

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE emp(
    id int,             -- Employee ID
    name varchar,    -- Employee name
    deptno int       -- Employee department ID
);

gaussdb=# CREATE TABLE dept(
    deptno int,      -- Department ID
    depname varchar -- Department name
);

gaussdb=# INSERT INTO emp VALUES (1, 'Joe', 10), (2, 'Scott', 20), (3, 'Ben', 999);   -- The department of Ben has not been confirmed yet. Therefore, the department ID is 999.
gaussdb=# INSERT INTO dept VALUES (10, 'hr'), (20, 'it'), (30, 'sal');               -- There is no employee in the sal department.
```

- INNER JOIN


```
gaussdb=# SELECT t1.id,t1.name,t2.depname FROM emp t1 JOIN dept t2 ON t1.deptno = t2.deptno;
 id | name  | depname
----+-------+---------
  1 | Joe   | hr
  2 | Scott | it
(2 rows)
```

- LEFT JOIN

```
gaussdb=# SELECT t1.id,t1.name,t2.depname FROM emp t1 LEFT JOIN dept t2 ON t1.deptno =
t2.deptno;
 id | name  | depname
----+-------+---------
  1 | Joe   | hr
  2 | Scott | it
  3 | Ben   |
(3 rows)
```

- RIGHT JOIN

```
gaussdb=# SELECT t1.id,t1.name,t2.depname FROM emp t1 RIGHT JOIN dept t2 ON t1.deptno =
t2.deptno;
 id | name  | depname
----+-------+---------
  1 | Joe   | hr
  2 | Scott | it
    |       | sal
(3 rows)
```

- FULL JOIN

```
gaussdb=# SELECT t1.id,t1.name,t2.depname FROM emp t1 FULL JOIN dept t2 ON t1.deptno =
t2.deptno;
 id | name  | depname
----+-------+---------
  1 | Joe   | hr
  2 | Scott | it
    |       | sal
  3 | Ben   |
(4 rows)

-- Delete.
gaussdb=# DROP TABLE emp,dept;
```

7.12.18.3.7 Subqueries

A query can be nested in another query, and its result is used as the data source or condition of the other query. Outer queries are also called parent queries, and inner queries are also called subqueries.

- Subqueries can be classified into single-row subqueries and multi-row subqueries based on the number of records returned.
- Subqueries can be classified into correlated subqueries and non-correlated subqueries based on whether a subquery is executed for multiple times.

Single-Row Subqueries

Single-row subquery operators include >=, >, <=, <, and <>.

```
-- Create a student table and insert data into the table.
gaussdb=# CREATE TABLE student(
    sid VARCHAR(5),      -- Student ID
    grade INT,           -- Grade
    name VARCHAR(20),    -- Name
    height INT           -- Height
);
gaussdb=# INSERT INTO student VALUES ('00001',1,'Scott',135),('00002',1,'Jack',95),('00003',1,'Ben',100);
gaussdb=# INSERT INTO student VALUES ('00004',2,'Henry',115),('00005',2,'Jordan',130),
('00006',2,'Bob',126);
gaussdb=# INSERT INTO student VALUES ('00007',3,'Bella',128),('00008',3,'Alicia',136);
```



```
-- Create a teacher table and insert data into the table.
gaussdb=# CREATE TABLE teacher (
    name VARCHAR(20),    -- Teacher name
    grade INT            -- Class
);
-- Insert data.
gaussdb=# INSERT INTO teacher VALUES ('Bill',1),('Sally',2),('Luke',3);

-- Query the students who are taller than Bella.
gaussdb=# SELECT * FROM student
WHERE height > (SELECT height FROM student WHERE name = 'Bella');
 sid  | grade |  name  | height
-------+-------+--------+--------
 00001 |     1 | Scott  |    135
 00005 |     2 | Jordan |    130
 00008 |     3 | Alicia |    136
(3 rows)
```

Multi-Row Subqueries

Multi-row subquery operators are as follows:

- IN: equal to any value in the list.
- ANY: used together with single-row comparison operators to compare with any value returned by the subquery.
- ALL: used together with single-row comparison operators to compare with all values returned by the subquery.
- SOME: another name for ANY and both operators have the same effect.

Example: Query the students of Sally and Luke.

```
gaussdb=# SELECT * FROM student t1 WHERE t1.grade IN (
    SELECT grade FROM teacher WHERE name = 'Sally' OR name = 'Luke'
);
 sid  | grade |  name  | height
-------+-------+--------+--------
 00004 |     2 | Henry  |    115
 00005 |     2 | Jordan |    130
 00006 |     2 | Bob    |    126
 00007 |     3 | Bella  |    128
 00008 |     3 | Alicia |    136
(5 rows)
```

Example: Query the students in grade 2 who are taller than any students in grade 3.

```
gaussdb=# SELECT * FROM student
WHERE grade = 2 AND
      height > ANY (SELECT height FROM student WHERE grade = 3);
 sid  | grade |  name  | height
-------+-------+--------+--------
 00005 |     2 | Jordan |    130
(1 row)

-- The query result is equivalent to that of the following query:
gaussdb=# SELECT * FROM student
WHERE grade = 2 AND
      height > (SELECT MIN(height) FROM student WHERE grade = 3);
```

Example: Query the students in grade 1 who are taller than all students in grade 2.

```
gaussdb=# SELECT * FROM student
WHERE grade = 1 AND
      height > ALL (SELECT height FROM student WHERE grade = 2);
```



```
 sid  | grade | name  | height
-------+-------+-------+--------
 00001 |     1 | Scott |    135
(1 row)

-- The query result is equivalent to that of the following query:
gaussdb=# SELECT * FROM student
WHERE grade = 1 AND
      height > (SELECT MAX(height) FROM student WHERE grade = 2);
```

Correlated Subqueries

Characteristics: Subqueries cannot run independently and are related to parent queries. Execute parent queries and then subqueries. Each time a parent query is executed, its subquery is recalculated.

Example: Query the students whose height is greater than the average height of the students in the class.

```
gaussdb=# SELECT * FROM student out
WHERE height > (SELECT AVG(height) FROM student
               WHERE grade = out.grade);
 sid  | grade |  name  | height
-------+-------+--------+--------
 00001 |     1 | Scott  |    135
 00005 |     2 | Jordan |    130
 00006 |     2 | Bob    |    126
 00008 |     3 | Alicia |    136
(4 rows)
```

Non-Correlated Subqueries

Characteristics: A subquery queries the value and then returns the result to the outer layer for query.

Example: Query the students whose height is greater than the average height of the students in the class.

```
gaussdb=# SELECT t1.*
FROM student t1,
    (SELECT grade, AVG(height) avg_hei FROM student GROUP BY grade) t2
WHERE t1.grade = t2.grade AND
      t1.height > t2.avg_hei;
 sid  | grade |  name  | height
-------+-------+--------+--------
 00001 |     1 | Scott  |    135
 00005 |     2 | Jordan |    130
 00006 |     2 | Bob    |    126
 00008 |     3 | Alicia |    136
(4 rows)
-- Delete.
gaussdb=# DROP TABLE student;
gaussdb=# DROP TABLE teacher;
```

7.12.18.3.8 Hierarchical Queries

Syntax

```
SELECT [level], column FROM table
START WITH condition
CONNECT BY[PRIOR column1 = column2|column1 = prior column2]
[ORDER SIBLINGS BY] [GROUP BY]
```


Parameters

- **level**: pseudocolumn. It is used to return the query level.
- **START WITH**: specifies the hierarchy, that is, the root row of the query.
- **CONNECT BY**: specifies the relationship between parent rows and child rows.
- **PRIOR**: specifies the parent level.
- **ORDER SIBLINGS BY**: specifies the sequence in the same level.

Examples

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE area (id INT,name VARCHAR(25),parent_id INT);
gaussdb=# INSERT INTO area VALUES (1,'China',NULL);
gaussdb=# INSERT INTO area VALUES (2,'Beijing',1);
gaussdb=# INSERT INTO area VALUES(3,'Chaoyang District',2);
gaussdb=# INSERT INTO area VALUES(4,'Shaanxi Province',1);
gaussdb=# INSERT INTO area VALUES(5,'Xi'an',4);
gaussdb=# INSERT INTO area VALUES(6,'Yanta District',5);
gaussdb=# INSERT INTO area VALUES(7,'Weiyang District',5);

-- Query the hierarchy.
gaussdb=# SELECT level, name FROM area
   START WITH (id = 1)
   CONNECT BY PRIOR id = parent_id;
 level |  name
-------+--------
     1 | China
     2 | Shaanxi Province
     3 | Xi'an
     4 | Weiyang District
     4 | Yanta District
     2 | Beijing
     3 | Chaoyang District
(7 rows)
```

- Use WHERE to remove a node (only Beijing).

```
-- Use WHERE to remove a node (only Beijing).
gaussdb=# SELECT level, name FROM area
   WHERE id <> 2
   START WITH (id = 1)
   CONNECT BY PRIOR id = parent_id;
 level |  name
-------+--------
     1 | China
     2 | Shaanxi Province
     3 | Xi'an
     4 | Weiyang District
     4 | Yanta District
     3 | Chaoyang District
(6 rows)
```

- Add a condition to the CONNECT BY PRIOR clause to remove a branch (Beijing and its sub-regions).

```
gaussdb=# SELECT level, name FROM area
   START WITH (id = 1)
   CONNECT BY PRIOR id = parent_id AND id <> 2;
 level |  name
-------+--------
     1 | China
     2 | Shaanxi Province
     3 | Xi'an
     4 | Weiyang District
     4 | Yanta District
(5 rows)

-- Delete.
gaussdb=# DROP TABLE area;
```


7.12.18.3.9 Compound Queries

A query that contains a compound operator is a compound query. All compound queries have the same priority. The number of columns and expressions in each query result in a set operation must be the same, and the types must be compatible.

Common set operations include:

- UNION: a union set of the two query result sets where duplicates are removed.
- UNION ALL: a union set of the two queries where the results of the two queries are simply combined.
- INTERSECT: intersection of two queries (only duplicated data is displayed).
- MINUS: difference of two query result sets. Only the data that exists in the first result set but does not exist in the second result set is displayed, and the data is sorted based on the results in the first column.

Examples

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE test1(c11 INT, c12 VARCHAR);
gaussdb=# INSERT INTO test1 VALUES (1,'a'),(2,'b'),(4,'d');

gaussdb=# CREATE TABLE test2(c21 INT, c22 VARCHAR);
gaussdb=# INSERT INTO test2 VALUES (1,'a'),(3,'c');
```

- UNION

```
gaussdb=# SELECT * FROM test1 UNION SELECT * FROM test2;
 c11 | c12
-----+-----
   1 | a
   4 | d
   2 | b
   3 | c
(4 rows)
```

- UNION ALL

```
gaussdb=# SELECT * FROM test1 UNION ALL SELECT * FROM test2;
 c11 | c12
-----+-----
   1 | a
   2 | b
   4 | d
   1 | a
   3 | c
(5 rows)
```

- INTERSECT

```
gaussdb=# SELECT * FROM test1 INTERSECT SELECT * FROM test2;
 c11 | c12
-----+-----
   1 | a
(1 row)
```

- MINUS

```
gaussdb=# SELECT * FROM test1 MINUS SELECT * FROM test2;
 c11 | c12
-----+-----
   4 | d
   2 | b
(2 rows)
-- Delete.
gaussdb=# DROP TABLE test1,test2;
```


7.12.18.3.10 Rows to Columns and Columns to Rows

- Rows to columns

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE test_p1(id INT, math INT, english INT);
gaussdb=# INSERT INTO test_p1 VALUES (1,84,78), (2,98,82), (3,68,59);
gaussdb=# SELECT * FROM test_p1;
 id | math | english
----+------+---------
  3 |   68 |      59
  1 |   84 |      78
  2 |   98 |      82
(3 rows)

-- Convert rows to columns.
gaussdb=# SELECT * FROM test_p1 UNPIVOT(score FOR class IN(math, english));
 id |  class  | score
----+---------+-------
  3 | MATH    |    68
  3 | ENGLISH |    59
  1 | MATH    |    84
  1 | ENGLISH |    78
  2 | MATH    |    98
  2 | ENGLISH |    82
(6 rows)

-- Delete.
gaussdb=# DROP TABLE test_p1;
```

- Columns to rows

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE test_p2(id INT, class VARCHAR(20), score INT);
gaussdb=# INSERT INTO test_p2 VALUES (1,'math',64), (1,'english',78);
gaussdb=# INSERT INTO test_p2 VALUES (2,'math',98), (2,'english',82);
gaussdb=# INSERT INTO test_p2 VALUES (3,'math',68), (3,'english',59);
gaussdb=# SELECT * FROM test_p2;
 id |  class  | score
----+---------+-------
  3 | math    |    68
  3 | english |    59
  1 | math    |    64
  1 | english |    78
  2 | math    |    98
  2 | english |    82
(6 rows)

-- Convert columns to rows.
gaussdb=# SELECT * FROM test_p2 PIVOT(MAX(score) FOR class IN('math','english'));
 id | 'math' | 'english'
----+--------+-----------
  3 |     68 |        59
  1 |     64 |        78
  2 |     98 |        82
(3 rows)

-- Delete.
gaussdb=# DROP TABLE test_p2;
```

7.12.18.4 SELECT INTO

Description

SELECT INTO creates a table based on the query result and inserts data retrieved by the query to the new table. Different from common SELECT, SELECT INTO does not return data to the client. The columns of the new table have the same names and data types as the output columns of SELECT.

Precautions

CREATE TABLE AS provides functions similar to SELECT INTO in functions and provides a superset of functions provided by SELECT INTO. You are advised to use CREATE TABLE AS, because SELECT INTO cannot be used in a stored procedure.

Syntax

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    { * | {expression [ [ AS ] output_name ]} [, ...] }
    INTO [ [ GLOBAL | LOCAL ] [ TEMPORARY | TEMP ] | UNLOGGED ] [ TABLE ] new_table
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY expression [, ...] ]
    [ HAVING condition [, ...] ]
    [ WINDOW {window_name AS ( window_definition )} [, ...] ]
    [ { UNION | INTERSECT | EXCEPT | MINUS } [ ALL | DISTINCT ] select ]
    [ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS { FIRST |
LAST } ]} [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
    [ {FOR { UPDATE | SHARE } [ OF table_name [, ...] ] [ NOWAIT |WAIT N]} [...] ];
```

Parameters

- **new_table**

  Specifies the name of the new table.

- **UNLOGGED**

  Specifies that the table is created as an unlogged table. Data written to unlogged tables is not written to the WALs, which makes them considerably faster than ordinary tables. However, an unlogged table is automatically truncated after a crash or unclean shutdown. The contents of an unlogged table are also not replicated to standby servers. Any indexes created on an unlogged table are automatically unlogged as well.

  - Usage scenario: Unlogged tables do not ensure data security. Users can back up data before using unlogged tables; for example, users should back up the data before a system upgrade.
  - Troubleshooting: If data is missing in the indexes of unlogged tables due to some unexpected operations such as an unclean shutdown, users should rebuild the indexes with errors.

- **GLOBAL | LOCAL**

  When creating a temporary table, you can specify the GLOBAL or LOCAL keyword before TEMP or TEMPORARY. If the GLOBAL keyword is specified, GaussDB creates a global temporary table. Otherwise, GaussDB creates a local temporary table.

- **TEMPORARY | TEMP**

  If TEMP or TEMPORARY is specified, the created table is a temporary table. Temporary tables are classified into global temporary tables and local temporary tables. If the GLOBAL keyword is specified when a temporary table is created, the table is a global temporary table. Otherwise, the table is a local temporary table.

  The metadata of the global temporary table is visible to all sessions. After the sessions end, the metadata still exists. The user data, indexes, and statistics of

a session are isolated from those of another session. Each session can only view and modify the data committed by itself. Global temporary tables can be created using ON COMMIT PRESERVE ROWS or ON COMMIT DELETE ROWS. The former one creates a session-level table, where user data is automatically cleared when a session ends; the latter creates a transaction-level table, where user data is automatically cleared when a commit or rollback operation is performed. If the **ON COMMIT** option is not specified during table creation, the session level is used by default. Different from local temporary tables, you can specify a schema that does not start with **pg_temp_** when creating a global temporary table.

A local temporary table is visible only in the current session and is automatically dropped at the end of the session. Therefore, you can create and use temporary tables in the current session as long as the connected database node in the session is normal. Temporary tables are created only in the current session. If a DDL statement involves operations on temporary tables, a DDL error will be generated. Therefore, you are advised not to perform operations on temporary tables in DDL statements. TEMP is equivalent to TEMPORARY.

NOTICE

- Local temporary tables are visible to the current session through the schema starting with **pg_temp_**. You are advised not to manually delete the schema starting with **pg_temp_** or **pg_toast_temp_**.
- If TEMPORARY or TEMP is not specified when you create a table and its schema is set to that starting with **pg_temp_** in the current session, the table will be created as a temporary table.
- If another session is using a global temporary table or index, the ALTER or DROP operation cannot be performed on it.
- The DDL of a global temporary table affects only the user data and indexes of the current session. For example, **TRUNCATE**, **REINDEX**, and **ANALYZE** are valid only for the current session.

NOTE

For details about other SELECT INTO parameters, see **Parameters** in "SELECT."

Examples

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE tbl_person (
  id      integer,
  name    varchar(20),
  sex     varchar(5) CHECK(sex ='Male' or sex = 'Female')
);
gaussdb=# INSERT INTO tbl_person VALUES (1, 'Bob', 'Male'),(2, 'Anne', 'Female'),(3, 'Jack', 'Male'),(4,
'Danny', 'Male'),(5, 'Alice', 'Female'),(6, 'Susan', 'Female');

-- Add information about all persons whose sex is male in the person table to the new table.
gaussdb=# SELECT * INTO tbl_man FROM tbl_person WHERE sex = 'Male';

-- Query data in the tbl_man table.
gaussdb=# SELECT * FROM tbl_man;
 id | name  | sex
----+-------+-----
```



```
 1 | Bob   | Male
 3 | Jack  | Male
 4 | Danny | Male
(3 rows)

-- Drop the table.
gaussdb=# DROP TABLE tbl_person, tbl_man;
```

Helpful Links

SELECT

7.12.18.5 SET

Description

Modifies a GUC parameter.

Precautions

Most GUC parameters can be modified by executing **SET**. Some parameters cannot be modified after a server or session starts.

Syntax

- Set the system time zone.
  ```
  SET [ SESSION | LOCAL ] TIME ZONE { timezone | LOCAL | DEFAULT };
  ```
- Set the schema of the table.
  ```
  SET [ SESSION | LOCAL ]
    {CURRENT_SCHEMA { TO | = } { schema | DEFAULT }
    | SCHEMA 'schema'};
  ```
- Set client encoding.
  ```
  SET [ SESSION | LOCAL ] NAMES {'charset_name' [COLLATE 'collation_name'] | DEFAULT};
  ```
- Set other GUC parameters.
  ```
  SET [ LOCAL | SESSION ]
    {config_parameter { { TO | = } { value | DEFAULT }
                      | FROM CURRENT }};
  ```
- Set parameters in mode B (**sql_compatibility** = **'B'**).
  ```
  SET [ SESSION | @@SESSION. | @@]
      {config_parameter = { expr | DEFAULT }};
  ```
- Set user-defined user variables.
  ```
  SET @var_name := expr [, @var_name := expr] ...
  SET @var_name = expr [, @var_name = expr] ...
  ```

Parameters

- **SESSION**

  Specifies that the specified parameters take effect for the current session. This is the default value if neither **SESSION** nor **LOCAL** appears.

  If **SET** or **SET SESSION** is executed within a transaction that is later aborted, the effects of the **SET** statement disappear when the transaction is rolled back. Once the surrounding transaction is committed, the effects will persist until the end of the session, unless overridden by another **SET**.

- **LOCAL**

Specifies that the specified parameters take effect for the current transaction. After **COMMIT** or **ROLLBACK**, the session-level setting takes effect again.

The effects of **SET LOCAL** last only till the end of the current transaction, whether committed or not. A special case is **SET** followed by **SET LOCAL** within a single transaction: the **SET LOCAL** value will be seen until the end of the transaction, but afterward (if the transaction is committed) the **SET** value will take effect.

- **TIME ZONE timezone**

  Specifies the local time zone for the current session.

  Value range: a valid local time zone. The corresponding GUC parameter is **TimeZone**. The default value is **PRC**.

- **CURRENT_SCHEMA schema**

  Specifies the current schema.

  Value range: an existing schema name If the schema name does not exist, the value of **CURRENT_SCHEMA** will be empty.

- **SCHEMA schema**

  Specifies the current schema. Here the schema is a string.

  Example: set schema 'public';

- **NAMES {'charset_name' [COLLATE 'collation_name'] | DEFAULT};**
- The COLLATE clause can be specified when the **b_format_version** is set to **'5.7'** and **b_format_dev_version** set to **'s2'** in B-compatible mode.

  Sets the client character encoding, character set of a constant string, collation, and character set of the returned result.

  It is equivalent to:

  ```
  set client_encoding = charset_name; set character_set_connection = charset_name;
  set collation_connection = collation_name; set character_set_results = charset_name;
  ```

  Value range: character sets and collations supported in B-compatible mode. Currently, the database character set must be specified for **charset_name**.

- The COLLATE clause cannot be specified in other scenarios.

  Specifies the client character encoding.

  It is equivalent to:

  ```
  set client_encoding to charset_name
  ```

  Value range: a valid character encoding name. The GUC parameter corresponding to this option is **client_encoding**. The default encoding is **UTF8**.

- **config_parameter**

  Specifies the name of a configurable GUC parameter. You can use **SHOW ALL** to view available GUC parameters.

  > NOTE
  >
  > Some parameters that viewed by **SHOW ALL** cannot be set by **SET**. For example, **max_datanodes**.

- **value**

  Specifies the new value of **config_parameter**. This parameter can be specified as string constants, identifiers, numbers, or comma-separated lists of these. **DEFAULT** can be written to indicate resetting the parameter to its default value.

- **SESSION | @@SESSION. | @@**

  The declared parameter takes effect in superuser or user mode, which can be determined by the context column in the pg_settings system view. If neither GLOBAL nor SESSION exists, SESSION is used as the default value. The value of **config_parameter** can be an expression.

  NOTE

  1. SET SESSION is supported only in mode B (**sql_compatibility** = **'B'**) and the value of **b_format_behavior_compat_options** is set to **'enable_set_variables'**.
  2. When @@config_parameter is used for operator calculation, use spaces to separate them. For example, in the **set @@config_parameter1=@@config_parameter1*2;** command, **=@@** is used as an operator. You can change **=@@** to **set @config_parameter1= @@config_parameter1 * 2**.

- **var_name**

  User-defined variable name. A variable name can contain only digits, letters, underscores (_), periods (.), and dollar signs ($). If a variable name is quoted using single or double quotation marks, other characters can be used, such as 'var_name', "var_name", and `var_name`.

  NOTE

  - User-defined variables can be set only in mode B (**sql_compatibility** = **'B'**) and the value of **b_format_behavior_compat_options** is set to **'enable_set_variables'**.
  - User-defined variables store only integers, floating point numbers, strings, bit strings, and NULL. The BOOLEAN, INT1, INT2, INT4, and INT8 types will be converted to the INT8 type, and the FLOAT4, FLOAT8, and NUMBERIC types will be converted to the FLOAT8 type for storage. Note that the precision of the floating-point type may be lost. The BIT type is stored in BIT, the VARBIT type is stored in VARBIT, and the NULL value is stored in NULL. If other types can be converted into character strings, they are converted into TEXT for storage.
  - When @var_name is used for operator calculation, use spaces to separate them. For example, in the **set @v1=@v2+1;** command, **=@** is used as an operator. You can change **=@** to **set @v1= @v2+1**.
  - When **sql_compatibility** is set to **'B'** and **b_format_behavior_compat_options** is set to **'enable_set_variables'**, for the original @ expr of the database (see **Numeric Operator**), there must be a space between @ and expr. Otherwise, @ will be parsed into a user-defined variable.
  - The value of an uninitialized variable is NULL.
  - Character strings stored in user-defined variables in the PREPARE statement support only the SELECT, INSERT, UPDATE, DELETE, adn MERGE syntax.
  - In consecutive value assignment scenarios, only **@var_name1 := @var_name2 :=... := expr and @var_name1 = @var_name2 :=... := expr** is supported. An equal sign (=) indicates a value assignment only when it is placed at the beginning, and other positions indicate comparison operators.

- **expr**

  Expression, which can be directly or indirectly converted to an integer, floating point, string, bit string, or NULL.

  NOTE

  Do not use functions that contain sensitive information (such as passwords) in character string expressions, such as encryption and decryption functions gs_encrypt and gs_decrypt, to prevent sensitive information leakage.

Examples

```
-- Set the search path of a schema.
gaussdb=# SET search_path TO tpcds, public;

-- Set the date style to the traditional POSTGRES style (date placed before month):
gaussdb=# SET datestyle TO postgres,dmy;

--Set user-defined variables.
gaussdb=# CREATE DATABASE user_var dbcompatibility 'b';
gaussdb=# \c user_var
user_var=# SET b_format_behavior_compat_options = enable_set_variables;
user_var=# SET @v1 := 1, @v2 := 1.1, @v3 := true, @v4 := 'dasda', @v5 := x'41';

-- Query user-defined variables.
user_var=# SELECT @v1, @v2, @v3, @v4, @v5, @v6, @v7;

-- Use user-defined variables.
user_var=# SET @sql = 'select 1';
user_var=# PREPARE stmt as @sql;
user_var=# EXECUTE stmt;
user_var=# \q

-- Delete the database.
gaussdb=# DROP DATABASE user_var;
```

```
-- Set B-compatible parameters.
gaussdb=# CREATE DATABASE test_set dbcompatibility 'B';
gaussdb=# \c test_set
test_set=# set b_format_behavior_compat_options = 'enable_set_variables';

-- Set session variables.
test_set=# SET @@codegen_cost_threshold = 10000;
test_set=# SET @@session.codegen_cost_threshold = @@codegen_cost_threshold * 2;

-- Set global variables.
test_set=# \q

-- Delete the database.
gaussdb=# DROP DATABASE test_set;
```

Helpful Links

RESET and **SHOW**

7.12.18.6 SET CONSTRAINTS

Description

SET CONSTRAINTS sets the behavior of constraint checking within the current transaction.

IMMEDIATE constraints are checked at the end of each statement. DEFERRED constraints are not checked until transaction commit. Each constraint has its own **IMMEDIATE** or **DEFERRED** mode.

Upon creation, a constraint is given one of three characteristics **DEFERRABLE INITIALLY DEFERRED**, **DEFERRABLE INITIALLY IMMEDIATE**, or **NOT DEFERRABLE**. The third class is always **IMMEDIATE** and is not affected by the **SET CONSTRAINTS** statement. The first two classes start every transaction in specified modes, but its behaviors can be changed within a transaction by SET CONSTRAINTS.

SET CONSTRAINTS with a list of constraint names changes the mode of just those constraints (which must all be deferrable). If multiple constraints match a name, the name is affected by all of these constraints. **SET CONSTRAINTS ALL** changes the modes of all deferrable constraints.

When **SET CONSTRAINTS** changes the mode of a constraint from **DEFERRED** to **IMMEDIATE**, any data that is modified at the end of the transaction is checked during the execution of the **SET CONSTRAINTS** statement. If any such constraint is violated, the **SET CONSTRAINTS** fails (and does not change the constraint mode). Therefore, **SET CONSTRAINTS** can be used to force checking of constraints to occur at a specific point in a transaction.

Precautions

SET CONSTRAINTS sets the behavior of constraint checking only within the current transaction. Therefore, if you execute this statement outside of a transaction block (START TRANSACTION/COMMIT pair), it will not appear to have any effect.

Syntax

```
SET CONSTRAINTS { ALL | { name } [, ...] } { DEFERRED | IMMEDIATE };
```

Parameters

- **name**

  Specifies the constraint name.

  Value range: an existing table name, which can be found in the system catalog **pg_constraint**.

- **ALL**

  Specifies all constraints.

- **DEFERRED**

  Specifies that constraints are not checked until transaction commit.

- **IMMEDIATE**

  Specifies that constraints are checked at the end of each statement.

Examples

```
-- Set that constraints are checked when a transaction is committed.
gaussdb=# SET CONSTRAINTS ALL DEFERRED;
```

7.12.18.7 SET ROLE

Description

Sets the current user identifier of the current session.

Precautions

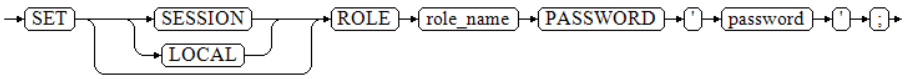
- Users of the current session must be members of specified **rolename**, but the system administrator can choose any roles.
- Executing this statement may add or restrict permissions of a user. If the role of a session user has the **INHERITS** attribute, it automatically has all

permissions of roles that SET ROLE enables the role to be. In this case, SET ROLE physically deletes all permissions directly granted to session users and permissions of its belonging roles and only leaves permissions of the specified roles. If the role of the session user has the **NOINHERITS** attribute, SET ROLE deletes permissions directly granted to the session user and obtains permissions of the specified role.

Syntax

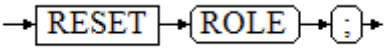
- Set the current user identifier of the current session.

```
SET [ SESSION | LOCAL ] ROLE role_name PASSWORD 'password';
```



- Reset the current user identifier to that of the current session.

```
RESET ROLE;
```



Parameters

- **SESSION**

  Specifies that the statement takes effect only for the current session. This parameter is used by default.

- **LOCAL**

  Specifies that the specified statement takes effect only for the current transaction.

- **role_name**

  Indicates the role name.

  Value range: a string. It must be an existing username in the database.

- **password**

  Specifies the password of a role. It must comply with the password convention.

  NOTE

  - The restrictions on using a ciphertext password are as follows:
    - An administrator cannot use a ciphertext password to switch to another administrator but to a user with lower permissions.
  - Ciphertext passwords are usually used in gs_dump and gs_dumpall export scenarios. In other scenarios, you are advised not to use ciphertext passwords directly.

- **RESET ROLE**

  Resets the current user identifier.

Examples

```
-- Query the current session user and the current user.
gaussdb=# SELECT SESSION_USER, CURRENT_USER;
 session_user | current_user
--------------+--------------
 omm          | omm
(1 row)
```

```
-- Create a role paul.
gaussdb=# CREATE ROLE paul PASSWORD '********';

-- Set the current user to paul.
gaussdb=# SET ROLE paul PASSWORD '********';

-- View the current session user and the current user.
gaussdb=> SELECT SESSION_USER, CURRENT_USER;
 session_user | current_user
--------------+--------------
 omm          | paul
(1 row)

-- Reset the current user.
gaussdb=> RESET ROLE;

-- Delete the user.
gaussdb=# DROP USER paul;
```

Helpful Links

7.13.18.8-SET SESSION AUTHORIZATION

7.12.18.8 SET SESSION AUTHORIZATION

Description

Sets the session user identifier and the current user identifier of the current SQL session to a specified user.

Precautions

The session identifier can be changed only when the initial session user has the system administrator rights. Otherwise, the system supports the statement only when the authenticated username is specified.

Syntax

- Set the session user identifier and the current user identifier of the current session.
  ```
  SET [ SESSION | LOCAL ] SESSION AUTHORIZATION role_name PASSWORD 'password';
  ```
- Reset the identifiers of the session and current users to the initially authenticated usernames.
  ```
  {SET [ SESSION | LOCAL ] SESSION AUTHORIZATION DEFAULT
    | RESET SESSION AUTHORIZATION};
  ```

Parameters

- **SESSION**

  Specifies that the specified parameters take effect for the current session.
- **LOCAL**

  Specifies that the specified statement takes effect only for the current transaction.
- **role_name**

  Username.

Value range: a string. It must be an existing username in the database.

- **password**

  Specifies the password of a role. It must comply with the password convention.

  NOTE

  The restrictions on using a ciphertext password are as follows:

  - An administrator cannot use a ciphertext password to switch to another administrator but to a user with lower permissions.

  Ciphertext passwords are usually used in gs_dump and gs_dumpall export scenarios. In other scenarios, you are advised not to use ciphertext passwords directly.

- **DEFAULT**

  Resets the identifiers of the session and current users to the initially authenticated usernames.

Examples

```
-- Query the session user and the current user.
gaussdb=# SELECT SESSION_USER, CURRENT_USER;
 session_user | current_user
--------------+--------------
 omm          | omm
(1 row)

-- Create a role paul.
gaussdb=# CREATE ROLE paul PASSWORD '********';

-- Set the current user to paul, and query the session user and current user.
gaussdb=# SET SESSION AUTHORIZATION paul PASSWORD '********';
gaussdb=> SELECT SESSION_USER, CURRENT_USER;
 session_user | current_user
--------------+--------------
 paul         | paul
(1 row)

-- Reset the session user and the current user.
gaussdb=> RESET SESSION AUTHORIZATION;
gaussdb=# SELECT SESSION_USER, CURRENT_USER;
 session_user | current_user
--------------+--------------
 omm          | omm
(1 row)

-- Delete the user.
gaussdb=# DROP USER paul;
```

Helpful Links

SET ROLE

7.12.18.9 SET TRANSACTION

Function

SET TRANSACTION sets characteristics of a transaction. Available transaction characteristics include the transaction separation level and transaction access mode (read/write or read only). You can set the local features of the current transaction, the default global transaction features inside a session, or the global transaction features among sessions of the current database.

Precautions

The current transaction characteristics must be set in a transaction, that is, **START TRANSACTION** or **BEGIN** must be executed before **SET TRANSACTION** is executed. Otherwise, the setting does not take effect. The settings of the global transaction features among sessions of the current database take effect after reconnection.

Syntax

Set the isolation level and access mode of the transaction.

```
{ SET [ LOCAL | SESSION | GLOBAL ] TRANSACTION|SET SESSION CHARACTERISTICS AS TRANSACTION }
  { ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE | REPEATABLE READ }
  | { READ WRITE | READ ONLY } };
```

Parameter Description

- **LOCAL**

  Specifies that the specified statement takes effect only for the current transaction.

- **SESSION**

  Specifies that the specified parameters take effect for the current session.

  The SET SESSION TRANSACTION statement takes effect only after **sql_compatibility** is set to **'B'** and **b_format_behavior_compat_options** is set to **set_session_transaction**. It is equivalent to SET SESSION CHARACTERISTICS AS TRANSACTION.

- **GLOBAL**

  Specifies that this command takes effect for global sessions of the current database.

  Application scope: This parameter takes effect in when **sql_compatibility** is set to **'B'**. It takes effect for subsequent sessions.

- **SESSION CHARACTERISTICS**

  Specifies that the specified parameters take effect for the current session.

- **ISOLATION LEVEL**

  Specifies the transaction isolation level that determines the data that a transaction can view if other concurrent transactions exist.

  NOTE

  - The isolation level cannot be changed after data is modified using SELECT, INSERT, DELETE, UPDATE, FETCH, or COPY in the current transaction.
  - The SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION LEVEL in a transaction block does not take effect for the current transaction. It takes effect only after the COMMIT operation is performed.

  Value range:

  - **READ COMMITTED**: Only submitted data is read. It is the default value.
  - **REPEATABLE READ**: Only the data committed before transaction start is read. Uncommitted data or data committed in other concurrent transactions cannot be read.
  - **SERIALIZABLE**: Currently, this isolation level is not supported in GaussDB. It is equivalent to **REPEATABLE READ**.

- **READ WRITE | READ ONLY**

  Specifies the transaction access mode (read/write or read only).

Examples

```
-- Create and switch to the test database.
gaussdb=# CREATE DATABASE mysql_compatible_db DBCOMPATIBILITY 'B';
gaussdb=# \c mysql_compatible_db

-- Start a transaction and set its isolation level to READ COMMITTED and access mode to READ ONLY.
gaussdb=# START TRANSACTION;
gaussdb=# SET LOCAL TRANSACTION ISOLATION LEVEL READ COMMITTED READ ONLY;
gaussdb=# COMMIT;

-- Set the transaction isolation level and read/write mode of the current session.
-- In mode B (sql_compatibility = 'B'), set b_format_behavior_compat_options to
set_session_transaction.
gaussdb=# SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
gaussdb=# SET SESSION TRANSACTION READ ONLY;

-- Set the transaction isolation level and read/write mode of all sessions of the current database
(sql_compatibility = 'B').
gaussdb=# SET GLOBAL TRANSACTION ISOLATION LEVEL READ COMMITTED;
gaussdb=# SET GLOBAL TRANSACTION READ ONLY;

-- Switch back to the initial database and delete the test database. Replace postgres with the actual
database name.
gaussdb=# \c postgres
gaussdb=# DROP DATABASE mysql_compatible_db;
```

7.12.18.10 SHOW

Function

SHOW shows the current value of a run-time parameter.

Syntax

```
SHOW
 {
   [VARIABLES LIKE] configuration_parameter |
   CURRENT_SCHEMA |
   TIME ZONE |
   TRANSACTION ISOLATION LEVEL |
   SESSION AUTHORIZATION |
   ALL
 };
```

Parameter Description

See **Parameters** in **RESET**.

Examples

```
-- Show the value of timezone.
gaussdb=# SHOW timezone;

-- Show all parameters.
gaussdb=# SHOW ALL;

-- Show all parameters whose names contain var.
gaussdb=# SHOW VARIABLES LIKE var;
```


Helpful Links

SET and **RESET**

7.12.18.11 SHOW EVENTS

Description

Displays basic information about all scheduled tasks in a specified schema.

Precautions

Operations related to scheduled tasks are supported only when **sql_compatibility** is set to **'B'**.

Syntax

```
SHOW EVENTS
   [{FROM | IN} schema_name]
   [LIKE 'pattern' | WHERE condition];
```

Parameters

- **{FROM | IN}**

  Specifies the schema to be queried. By default, the current schema is queried.

- **LIKE**

  Matches a scheduled task by name. If this parameter is not specified, all scheduled tasks in the current schema are printed.

- **WHERE**

  Forms an expression for row selection to narrow down the query range of **SHOW EVENTS**. **condition** indicates any expression that returns a value of Boolean type. Rows that do not meet this condition will not be retrieved.

Examples

```
-- Create and switch to the test database.
gaussdb=# CREATE DATABASE test_event WITH DBCOMPATIBILITY = 'b';
test_event=# \c test_event

-- Create a schema.
test_event=# CREATE SCHEMA test;

-- Create a table.
test_event=# CREATE TABLE t_ev(num int);

-- Create a scheduled task in the current schema by default.
test_event=# CREATE EVENT IF NOT EXISTS event_e2 ON SCHEDULE EVERY 1 minute DO insert into t_ev
values(1);

-- Create a scheduled task in the test schema.
test_event=# CREATE EVENT IF NOT EXISTS test.event_e3 ON SCHEDULE EVERY 1 minute DO insert into
t_ev values(2);

-- View the scheduled tasks.
test_event=# SHOW EVENTS;
job_name | schema_name | log_user | priv_user | job_status |        start_date         |       interval       |
end_date       | enable |                failure_msg
----------+-------------+----------+-----------+------------+---------------------------+----------------------
```



```
+---------------------+--------+-------------------------------
event_e2  | public      | omm       | omm       | s          | 2023-11-28 10:13:48.675215 | interval '1'  minute |
3999-12-31 16:00:00 | t      |
(1 rows)

-- View the scheduled task in the test schema.
test_event=# SHOW EVENTS FROM test;
job_name | schema_name | log_user | priv_user | job_status |         start_date          |       interval       |
end_date       | enable |                failure_msg
----------+-------------+----------+-----------+------------+-----------------------------+----------------------
+---------------------+--------+-------------------------------event_e3 | test        | omm       | omm       | s          |
2023-11-28 10:29:42.327827 | interval '1'  minute | 3999-12-31 16:00:00 | t      |
(1 row)

-- Delete the scheduled tasks.
test_event=# DROP EVENT event_e2;
test_event=# DROP EVENT test.event_e3;

-- Delete the table.
test_event=# DROP TABLE t_ev;

-- Delete a schema.
test_event=# DROP SCHEMA test;

-- Switch back to the initial database and delete the test database. Replace postgres with the actual
database name.
test_event=# \c postgres
gaussdb=# DROP DATABASE test_event;
```

Helpful Links

ALTER EVENT, **CREATE EVENT**, and **DROP EVENT**

7.12.18.12 SHUTDOWN

Function

SHUTDOWN shuts down the currently connected database node.

Precautions

Only the administrator can run this command.

Syntax

```
SHUTDOWN [FAST | IMMEDIATE];
```

Parameter Description

- **FAST | IMMEDIATE**

  FAST: Rolls back all active transactions, forcibly disconnects the client, and shuts down the database node without waiting for the client to disconnect.

  IMMEDIATE: Shuts down the server forcibly. Fault recovery will occur on the next startup.

  If the shutdown mode is not specified, the default value **FAST** is used.

Examples

```
-- Shut down the current database node.
gaussdb=# SHUTDOWN;
```



```
-- Shut down the current database node in fast mode.
gaussdb=# SHUTDOWN FAST;
```

7.12.18.13 SNAPSHOT

Description

Controls data in a unified manner for multiple users.

Precautions

- The GUC parameter **db4ai_snapshot_mode** classifies the snapshot storage models into MSS and CSS. The GUC parameter **db4ai_snapshot_version_delimiter** specifies the version separator and its default value is **@**. The GUC parameter **db4ai_snapshot_version_separator** specifies the sub-version separator and its default value is **.**.
- When the incremental storage mode is used for snapshots, the snapshots are dependent on each other. Snapshots must be deleted in the dependency sequence.
- The snapshot feature is used to maintain data between team members, involving data transcription between administrators and common users. Therefore, the snapshot feature is not supported in separation of duty (**enableSeparationOfDuty** is set to **ON**) scenarios.
- When you need a stable and available snapshot for tasks such as AI training, you need to publish the snapshot.

Syntax

1. Create a snapshot.

   You can use the CREATE SNAPSHOT... AS and CREATE SNAPSHOT... FROM statements to create a data table snapshot.

   – CREATE SNAPSHOT AS

   ```
   CREATE SNAPSHOT qualified_name [@ [version]]
     [COMMENT IS comment_item]
     AS query;
   ```

   – CREATE SNAPSHOT FROM

   ```
   CREATE SNAPSHOT qualified_name [@ [version]]
     FROM @ version
     [COMMENT IS comment_item]
       USING (
       { INSERT [INTO SNAPSHOT] insert_item
         | UPDATE [SNAPSHOT] [AS alias] SET set_item [FROM from_item] [WHERE where_item]
         | DELETE [FROM SNAPSHOT] [AS alias] [USING using_item] [WHERE where_item]
         | ALTER [SNAPSHOT] { ADD and_item | DROP drop_item } [, ...]
       } [; ...]
     );
   ```

2. Delete a snapshot.

   PURGE SNAPSHOT

   ```
   PURGE SNAPSHOT qualified_name @ version;
   ```

3. Sample snapshots.

   SAMPLE SNAPSHOT

   ```
   SAMPLE SNAPSHOT qualified_name @ version
     [STRATIFY BY attr_list]
     { AS alias AT RATIO num [COMMENT IS comment_item] } [, ...]
   ```


4. Publish snapshots.

   PUBLISH SNAPSHOT
   ```
   PUBLISH SNAPSHOT qualified_name @ version;
   ```

5. Archive snapshots.

   ARCHIVE SNAPSHOT
   ```
   ARCHIVE SNAPSHOT qualified_name @ version;
   ```

6. Query snapshots.
   ```
   SELECT * FROM DB4AISHOT ( qualified_name @ version);
   ```

Parameters

- **qualified_name**

  Name of the snapshot to be created

  Value range: a string. It must comply with the **naming convention**.

- **version**

  (Optional) Version number of a snapshot. This parameter is optional. The system automatically extends the sequence number.

  Value range: string, consisting of numbers and separators.

- **comment_item**

  Specifies the comment to be added.

  Value range: a string. It must comply with the **naming convention**.

- **insert_item**

  Specifies the name of the object to be inserted.

  Value range: a string. It must comply with the **naming convention**.

- **alias**

  Specifies the alias of the current object.

  Value range: a string. It must comply with the **naming convention**.

- **set_item**

  Specifies the name of the object on which the operation is performed.

  Value range: a string. It must comply with the **naming convention**.

- **from_item**

  Specifies the name of the query source object connected.

  Value range: a string. It must comply with the **naming convention**.

- **where_item**

  The return value is any expression of the Boolean type.

  Value range: a string. It must comply with the **naming convention**.

- **using_item**

  Specifies the name of the object used for connection.

  Value range: a string. It must comply with the **naming convention**.

- **and_item**

  Specifies the name of the object to be processed in parallel.

  Value range: a string. It must comply with the **naming convention**.

- **drop_item**

Specifies the name of the object to be dropped.

Value range: a string. It must comply with the **naming convention**.

- **attr_list**

  Specifies the list of target objects.

  Value range: a string. It must comply with the **naming convention**.

- num

  Specifies the ratio.

  Its value is a digit.

Examples

```
-- Create a data table.
gaussdb=# CREATE TABLE t1 (id int, name varchar);

-- Insert data.
gaussdb=# INSERT INTO t1 VALUES (1, 'zhangsan');
gaussdb=# INSERT INTO t1 VALUES (2, 'lisi');
gaussdb=# INSERT INTO t1 VALUES (3, 'wangwu');
gaussdb=# INSERT INTO t1 VALUES (4, 'lisa');
gaussdb=# INSERT INTO t1 VALUES (5, 'jack');

-- Create a snapshot.
gaussdb=# CREATE SNAPSHOT s1@1.0 comment is 'first version' AS SELECT * FROM t1;

-- Create snapshots in iteration mode.
gaussdb=# CREATE SNAPSHOT s1@2.0 FROM @1.0 comment is 'inherits from @1.0' USING (INSERT
VALUES(6, 'john'), (7, 'tim'); DELETE WHERE id = 1);

-- View snapshot content.
gaussdb=#SELECT * FROM DB4AISHOT(s1@1.0);

-- Sample snapshots.
gaussdb=# SAMPLE SNAPSHOT s1@2.0 stratify by name as nick at ratio .5;

-- Delete snapshots.
gaussdb=# PURGE SNAPSHOT s1@2.0;
gaussdb=# PURGE SNAPSHOT s1nick@2.0;
gaussdb=# PURGE SNAPSHOT s1@1.0;

-- Delete the table.
gaussdb=# DROP TABLE t1;
```

7.12.18.14 START TRANSACTION

Description

START TRANSACTION starts a transaction. If the isolation level or read/write mode is specified, a new transaction will have those characteristics. You can also specify them using **SET TRANSACTION**.

Syntax

Format 1: START TRANSACTION

```
START TRANSACTION
 [
  {
    ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE | REPEATABLE READ }
    | { READ WRITE | READ ONLY }
```



```
    } [ ...]
 ];
```

Format 2: BEGIN

```
BEGIN [ WORK | TRANSACTION ]
 [
  {
    ISOLATION LEVEL { READ COMMITTED | SERIALIZABLE | REPEATABLE READ }
    | { READ WRITE | READ ONLY }
   } [ ...]
 ];
```

Parameters

- **WORK | TRANSACTION**

  Specifies the optional keyword in BEGIN format without functions.

- **ISOLATION LEVEL**

  Specifies the transaction isolation level that determines the data that a transaction can view if other concurrent transactions exist.

  NOTE

  The isolation level of a transaction cannot be reset after the first clause (SELECT, INSERT, DELETE, UPDATE, FETCH, or COPY) for modifying data is executed in the transaction.

  Value range:

  - **READ COMMITTED**: Only submitted data is read. It is the default value.
  - **REPEATABLE READ**: Only the data committed before transaction start is read. Uncommitted data or data committed in other concurrent transactions cannot be read.
  - **SERIALIZABLE**: Currently, GaussDB does not support this isolation level. Setting this isolation level is equivalent to **REPEATABLE READ**.

- **READ WRITE | READ ONLY**

  Specifies the transaction access mode (read/write or read only).

Examples

```
-- Create a schema.
gaussdb=# CREATE SCHEMA tpcds;

-- Create the tpcds.reason table.
gaussdb=# CREATE TABLE tpcds.reason (c1 int, c2 int);

-- Start a transaction in default mode.
gaussdb=# START TRANSACTION;
gaussdb=# SELECT * FROM tpcds.reason;
gaussdb=# END;

-- Start a transaction in default mode.
gaussdb=# BEGIN;
gaussdb=# SELECT * FROM tpcds.reason;
gaussdb=# END;

-- Start a transaction with the isolation level being READ COMMITTED and the access mode being READ WRITE:
gaussdb=# START TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
gaussdb=# SELECT * FROM tpcds.reason;
gaussdb=# COMMIT;
```



```
-- Delete the tpcds.reason table.
gaussdb=# DROP TABLE tpcds.reason;

-- Delete the schema.
gaussdb=# DROP SCHEMA tpcds;
```

Helpful Links

COMMIT | END, **ROLLBACK**, and **SET TRANSACTION**

7.12.19 T

7.12.19.1 TIMECAPSULE TABLE

Description

The **TIMECAPSULE TABLE** statement restores a table to an earlier state in the event of human or application errors.

The table can flash back to a past point in time, depending on the old version of the data stored in the system. In addition, GaussDB cannot restore a table to an earlier state through DDL operations that has changed the structure of the table.

Precautions

- The **TIMECAPSULE TABLE** statement can be used to flash back the data of the old version or the data from the recycle bin.
  - **TO TIMECAPSULE** and **TO CSN** can flash a table back to an earlier version. Currently, only the Ustore engine is supported.
  - The recycle bin records the objects dropped or truncated by running **DROP** and **TRUNCATE**. **TO BEFORE DROP** and **TO BEFORE TRUNCATE** can flash data back from the recycle bin. Currently, the Ustore and Astore engines are supported.
- The following object types do not support flashback: system catalogs, DFS tables, global temporary tables, local temporary tables, unlogged tables, sequence tables, encrypted tables, and hash bucket tables.
- If a statement (DDL, DCL, or VACUUM FULL) that modifies the table structure or affects physical storage is executed between the flashback point and the current point, the flashback fails.
- To run **DROP**, you must have the CREATE or USAGE permission on the schema to which the junk object belongs, and you must be the owner of the schema or the owner of the junk object.

  To run **TRUNCATE**, you must have the CREATE or USAGE permission on the schema to which the junk object belongs, and you must be the owner of the schema or the junk object. In addition, you must have the TRUNCATE permission on the junk object.
- Scenarios or tables that do not support DROP or TRUNCATE:
  - Scenario where the recycle bin is disabled (**enable_recyclebin** is set to **off**)

- Scenario where the system is in the maintenance state (**xc_maintenance_mode** is set to **on**) or is upgraded from an unsupported baseline version to a supported version
- Scenario where multiple objects are deleted (The **DROP** or **TRUNCATE TABLE** command is executed to delete multiple objects at the same time.)
- System catalogs, DFS tables, global temporary tables, local temporary tables, unlogged tables, sequence tables, encrypted tables, and hash bucket tables
- Objects in the recycle bin are cleared and do not support DROP or TRUNCATE FLASHBACK. The **recyclebin_retention_time** parameter specifies the retention period of objects in the recycle bin.
- The recycle bin does not support write operations such as DML, DCL, and DDL, and does not support DQL query operations.
- If a statement (such as DDL, DCL, VACUUM FULL, and partition adding/deleting/splitting/merging) that modifies the table structure or affects physical files are executed between the TRUNCATE TABLE and TRUNCATE flashback operations, the flashback fails.
- When a table is entirely deleted or truncated, partitions are moved to the recycle bin along with the entire table. A single deleted partition cannot be moved to the recycle bin. This avoids data consistency damage.
- If the object on which the table depends is an external object (for example, a composite object or a user-defined object), the table is physically deleted and is not moved to the recycle bin.

Syntax

```
TIMECAPSULE TABLE [schema.]table_name TO { CSN expr | TIMESTAMP expr | BEFORE { DROP [RENAME TO table_name] | TRUNCATE } };
```

Parameters

- **schema**

  Specifies a schema containing the table to be flashed back. If this parameter is not specified, the current schema is used.

- **table_name**

  Specifies a table name.

- **TO CSN**

  Specifies the CSN corresponding to the time point when the table is to be flashed back. *expr* must be a number representing a valid CSN.

- **TO TIMESTAMP**

  Specifies a timestamp value corresponding to the point in time to which you want to flash back the table. The result of *expr* must be a valid past timestamp (convert a string to a time type using the **TO_TIMESTAMP** function). The table will be flashed back to a time within approximately 3 seconds of the specified timestamp.

  Note: When the flashback point is too old, the old version cannot be obtained because it is recycled. As a result, the flashback fails and the error message "Restore point too old" is displayed.

- **TO BEFORE DROP**

  Retrieves dropped tables and their subobjects from the recycle bin.

  You can specify either the original user-specified name of the table or the system-generated name assigned to the object when it was deleted.

  - System-generated recycle bin object names are unique. Therefore, if you specify the system-generated name, the database retrieves that specified object. To see the content in your recycle bin, run **SELECT * FROM gs_recyclebin;**.
  - If you specify the user-specified name and the recycle bin contains more than one object of that name, the database retrieves the object that was moved to the recycle bin most recently. If you want to retrieve an older version of the table, then do one of these things:
    - Specify the system-generated recycle bin name of the table you want to retrieve.
    - Run the **TIMECAPSULE TABLE ... TO BEFORE DROP** statement until you retrieve the table you want.
  - When a dropped table is restored, only the base table name is restored, and the names of other subobjects remain the same as those in the recycle bin. You can run the DDL command to manually change the names of subobjects as required.
  - The recycle bin does not support write operations such as DML, DCL, and DDL, and does not support DQL query operations (supported in later versions).
  - The **recyclebin_retention_time** parameter has been set for specifying the retention period of objects in the recycle bin. The objects will be automatically deleted after the retention period expires.

- **RENAME TO**

  Specifies a new name for the table retrieved from the recycle bin.

- **TRUNCATE**

  Flashes back to the point in time before the TRUNCATE operation.

Examples

- **Flashing data back to a specified time**

  You need to set the **undo_retention_time** parameter to set the retention period of old undo logs.

  Contact the administrator for information about how to use the parameter.

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE tbl_test(c1 int, c2 int);
gaussdb=# INSERT INTO tbl_test VALUES (1,1),(2,2),(3,3);

-- Query the current time and next_csn on all global nodes.
gaussdb=# SELECT now();
              now
-------------------------------
 2023-11-27 17:06:34.840698+08
(1 row)

gaussdb=# SELECT int8in(xidout(next_csn)) FROM gs_get_next_xid_csn();
 int8in
--------
```



```
 25391
(6 row)

-- Modify data.
gaussdb=# UPDATE tbl_test SET c1=111, c2=222 WHERE c1=1;

-- Query data.
gaussdb=# SELECT * FROM tbl_test;
 c1  | c2
-----+-----
 111 | 222
   2 |   2
   3 |   3
(3 rows)
-- Flash back the data to the time before the modification. Replace the time based on the actual
situation.
gaussdb=# TIMECAPSULE table tbl_test TO TIMESTAMP to_timestamp('2023-11-27
17:06:34.840698','YYYY-MM-DD HH24:MI:SS.FF');

-- You can also use the following SQL statements:
gaussdb=# TIMECAPSULE table tbl_test TO CSN 25391;
gaussdb=# SELECT * FROM tbl_test;
 c1 | c2
----+----
  2 |  2
  3 |  3
  1 |  1
(3 rows)

-- Delete.
gaussdb=# DROP TABLE tbl_test;
```

- **Flashing back data from the recycle bin**

  Prerequisites:

  – The **enable_recyclebin** parameter has been enabled to enable the recycle bin. Contact the administrator for details about how to use the parameter.

  – The **recyclebin_retention_time** parameter has been set for specifying the retention period of objects in the recycle bin. The objects will be automatically deleted after the retention period expires. Contact the administrator for details about how to use the parameter.

```
-- Create a table and insert data into the table.
gaussdb=# CREATE TABLE tbl_test1(c1 int, c2 varchar(10));
gaussdb=# INSERT INTO tbl_test1 VALUES (1,'AAA'),(2,'BBB');

-- Delete the table.
gaussdb=# DROP TABLE tbl_test1;

-- Flash back to the time before the table is deleted.
gaussdb=# TIMECAPSULE TABLE tbl_test1 TO BEFORE DROP;

-- Query data.
gaussdb=# SELECT * FROM tbl_test1;
 c1 | c2
----+-----
  1 | AAA
  2 | BBB
(2 rows)

-- Delete a table. (The PURGE parameter is added to delete the table data from the recycle bin.)
gaussdb=# DROP TABLE tbl_test1 PURGE;
```


7.12.19.2 TRUNCATE

Description

TRUNCATE quickly removes all rows from a database table.

It has the same effect as an unqualified DELETE on each table, but TRUNCATE is faster because it does not actually scan the tables. This is most useful on large tables.

Precautions

- TRUNCATE TABLE has the same function as a DELETE statement with no WHERE clause, emptying a table.
- TRUNCATE TABLE uses less system and transaction log resources as compared with DELETE.
  - DELETE deletes a row each time, and records the deletion of each row in the transaction log.
  - TRUNCATE TABLE deletes all rows in a table by releasing the data page storing the table data, and records the releasing of the data page only in the transaction log.
- The differences between TRUNCATE, DELETE, and DROP are as follows:
  - TRUNCATE TABLE deletes content, releases space, but does not delete definitions.
  - DELETE TABLE deletes content, but does not delete definitions nor release space.
  - DROP TABLE deletes content and definitions, and releases space.

Syntax

- Delete data from a table.

```
TRUNCATE [ TABLE ] [ ONLY ] {table_name [ * ]} [, ... ]
    [ CONTINUE IDENTITY ] [ CASCADE | RESTRICT] [ PURGE ]};
```

- Truncate the data in a partition.

```
ALTER TABLE [ IF EXISTS  ] { [ ONLY  ] table_name
                        | table_name *
                        | ONLY ( table_name )  }
   TRUNCATE PARTITION { partition_name
                    | FOR (  partition_value  [, ...] )  } [ UPDATE GLOBAL INDEX ];
```

Parameters

- **ONLY**

  If **ONLY** is specified, only the specified table is cleared. If **ONLY** is not specified, the table and all its subtables (if any) are cleared.

- **table_name**

  Specifies the name (optionally schema-qualified) of the target table.

  Value range: an existing table name

- **CONTINUE IDENTITY**

  Does not change the values of sequences. This is the default action.

- **CASCADE | RESTRICT**

  - **CASCADE**: Clears all tables that are added to a group.
  - **RESTRICT** (default): refuses to truncate if any of the tables have foreign-key references from tables that are not listed in the command.
- **PURGE**

  Purges table data in the recycle bin by default.
- **partition_name**

  Specifies the partition in the target partitioned table.

  Value range: an existing table name
- **partition_value**

  Specifies the value of the specified partition key.

  The value specified by **PARTITION FOR** can uniquely identify a partition.

  Value range: value range of the partition key for the partition to be renamed

NOTICE

When the **PARTITION FOR** clause is used, the entire partition where **partition_value** is located is cleared.

- **UPDATE GLOBAL INDEX**

  If this parameter is used, all global indexes in the partitioned table are updated to ensure that data can be queried correctly using global indexes. If this parameter is not used, all global indexes in the partitioned table will become invalid.

Examples

- Clear table data.

```
-- Create the reason table.
gaussdb=# CREATE TABLE reason (r_reason_sk int,r_reason_id varchar(16),r_reason_desc
varchar(100));

-- Insert multiple records into the table.
gaussdb=# INSERT INTO reason values(1,'AAAAAAAABAAAAAAA','reason 1'),
(5,'AAAAAAAABAAAAAAA','reason 2'),
                                  (15,'AAAAAAAABAAAAAAA','reason 3'), (25,'AAAAAAAABAAAAAAA','reason
4'),
                                  (35,'AAAAAAAABAAAAAAA','reason 5'), (45,'AAAAAAAACAAAAAAA','reason
6');

-- Check the table information. The table size is about 8 KB.
gaussdb=# \d+
                                          List of relations
 Schema |  Name  | Type  | Owner |    Size    |                          Storage                          |
Description
--------+--------+-------+-------+------------+-----------------------------------------------------------
+-------------
 public | reason | table | omm1  | 8192 bytes |
{orientation=row,compression=no,storage_type=USTORE,segment=off} |
(1 row)

-- Run the DELETE statement without the WHERE condition to clear data in the table and check the
table size.
gaussdb=# DELETE FROM reason;
gaussdb=# \d+
                                          List of relations
```



```
 Schema |  Name  | Type  | Owner |    Size    |                          Storage                          |
Description
--------+--------+-------+-------+------------+-----------------------------------------------------------
+-------------
 public | reason | table | omm1  | 8192 bytes |
{orientation=row,compression=no,storage_type=USTORE,segment=off} |
(1 row)

-- Run the TRUNCATE statement to clear the reason table and check the table size.
gaussdb=# TRUNCATE TABLE reason;
gaussdb=# \d+
                                        List of relations
 Schema |  Name  | Type  | Owner |  Size   |                          Storage                          |
Description
--------+--------+-------+-------+---------+-----------------------------------------------------------
+-------------
 public | reason | table | omm1  | 0 bytes |
{orientation=row,compression=no,storage_type=USTORE,segment=off} |
(1 row)

-- Delete the table.
gaussdb=# DROP TABLE reason;
```

- Clear data of a partitioned table.

```
-- Create a partitioned table.
gaussdb=# CREATE TABLE reason_p(
    r_reason_sk integer,
    r_reason_id character(16),
    r_reason_desc character(100)
)PARTITION BY RANGE (r_reason_sk)(
    partition p_05_before values less than (05),
    partition p_15 values less than (15),
    partition p_25 values less than (25),
    partition p_35 values less than (35),
    partition p_45_after values less than (MAXVALUE)
);

-- Insert data.
gaussdb=# INSERT INTO reason_p values(1,'AAAAAAAABAAAAAAA','reason 1'),
(5,'AAAAAAAABAAAAAAA','reason 2'),
                                (15,'AAAAAAAABAAAAAAA','reason 3'), (25,'AAAAAAAABAAAAAAA','reason
4'),
                                (35,'AAAAAAAABAAAAAAA','reason 5'), (45,'AAAAAAAACAAAAAAA','reason
6');

-- Clear the p_05_before partition.
gaussdb=# ALTER TABLE reason_p TRUNCATE PARTITION p_05_before UPDATE GLOBAL INDEX;

-- Clear the p_15 partition.
gaussdb=# ALTER TABLE reason_p TRUNCATE PARTITION for (13) UPDATE GLOBAL INDEX;

-- Clear the partitioned table.
gaussdb=# TRUNCATE TABLE reason_p;

-- Delete the reason_p table.
gaussdb=# DROP TABLE reason_p;
```

7.12.20 U

7.12.20.1 UPDATE

Description

Updates specified columns in all rows that meet the specified conditions. You can use WHERE to declare conditions. All columns specified by the SET clause will be updated. Other columns retain their original values.

Precautions

- The owner of a table, users granted with the UPDATE permission on the table, or users granted with the UPDATE ANY TABLE permission can update data in the table. When the separation of duties is disabled, system administrators have this permission by default.
- You must have the SELECT permission on all tables involved in the expressions or conditions.
- The generated column cannot be directly written. In the UPDATE statement, values cannot be specified for generated columns, but the keyword DEFAULT can be specified.
- The multi-table update syntax does not apply to views and tables containing RULE.
- For the UPDATE statement whose subquery is a stream plan, the same row cannot be concurrently updated.
- You cannot modify the database character encoding by updating a system catalog. Such operation will cause exceptions in existing data or other operations. If you need to change the character encoding of a database, follow the database switching process to migrate data.

Syntax

```
Update a single table:
[ WITH [ RECURSIVE ] with_query [, ...] ]
UPDATE [/*+ plan_hint */] [ ONLY ] {table_name [ partition_clause ] | subquery | view_name} [ * ] [ [ AS ]
alias ]
SET {column_name = { expression | DEFAULT }
   |( column_name [, ...] ) = {( { expression | DEFAULT } [, ...] ) |sub_query }}[, ...]
   [ FROM from_list] [ WHERE condition | WHERE CURRENT OF cursor_name ]
   [ ORDER BY {expression [ [ ASC | DESC | USING operator ]
   [ LIMIT { count } ]
   [ RETURNING {*
            | {output_expression [ [ AS ] output_name ]} [, ...] }];

Update multiple tables:
[ WITH [ RECURSIVE ] with_query [, ...] ]
UPDATE [/*+ plan_hint */] table_list
SET {column_name = { expression | DEFAULT }
   |( column_name [, ...] ) = {( { expression | DEFAULT } [, ...] ) |sub_query }}[, ...]
   [ FROM from_list] [ WHERE condition ];

where sub_query can be:
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
{ * | {expression [ [ AS ] output_name ]} [, ...] }
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY grouping_element [, ...] ]
[ HAVING condition [, ...] ]
[ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS { FIRST |
LAST } ]} [, ...] ]
[ LIMIT { [offset,] count | ALL } ]
```

- The subquery **with_query** is as follows:

  ```
  with_query_name [ ( column_name [, ...] ) ]
     AS [ [ NOT ] MATERIALIZED ] ( {select | values | insert | update | delete} )
  ```

Parameters

- **WITH [RECURSIVE] with_query [, ...]**

  Specifies one or more subqueries that can be referenced by name in the main query, which is equivalent to a temporary table. This subquery statement

structure is called the common table expression (CTE) structure. When this structure is used, the execution plan contains the CTE SCAN content.

If RECURSIVE is specified, it allows a SELECT subquery to reference itself by name.

Format of **with_query**:

```
with_query_name [ ( column_name [, ...] ) ] AS [ [ NOT ] MATERIALIZED ] ( {select | values | insert | update | delete} )
```

- **with_query_name** specifies the name of the result set generated by a subquery. Such names can be used to access the result sets of subqueries in a query.
- **column_name** specifies the column name displayed in the subquery result set.
- Each subquery can be a SELECT, VALUES, INSERT, UPDATE, or DELETE statement.
- You can use MATERIALIZED or NOT MATERIALIZED to modify the CTE.
  - If **MATERIALIZED** is specified, the WITH query will be materialized, and a copy of the subquery result set is generated. The copy is directly queried at the reference point. Therefore, the WITH subquery cannot be jointly optimized with the SELECT statement trunk (for example, predicate pushdown and equivalence class transfer). In this scenario, you can use **NOT MATERIALIZED** for modification. If the WITH query can be executed as a subquery inline, the preceding optimization can be performed.
  - If the user does not explicitly declare the materialized attribute, comply with the following rules: If the CTE is referenced only once in the SELECT statement trunk to which it belongs and semantically supports inline execution, it will be rewritten as subquery inline execution. Otherwise, the materialized execution will be performed in CTE Scan mode.

- **plan_hint**

  Follows the UPDATE keyword in the **/*+ */** format. It is used to optimize the plan of an UPDATE statement block. For details, see **Hint-based Tuning**. In each statement, only the first **/*+** *plan_hint* ***/** comment block takes effect as a hint. Multiple hints can be written.

- **table_name**

  Specifies the name (optionally schema-qualified) of the table to be updated. If ONLY is specified before the table name, only matched rows in the table are updated. If it is not specified, any matching rows inherited from the table are also updated.

  Value range: an existing table name

  NOTE

  You can use database links to perform operations on remote tables. For details, see **DATABASE LINK**.

- **subquery**

  Specifies the subquery to be updated. When a subquery is updated, the subquery is regarded as a temporary view. The **CHECK OPTION** option can be added to the end of the subquery.


```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [/*+ plan_hint */] [ ALL ]
{ * | {expression [ [ AS ] output_name ]} [, ...] }
[ into_option ]
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ [ START WITH condition ] CONNECT BY [NOCYCLE] condition [ ORDER SIBLINGS BY expression ] ]
[ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS
{ FIRST | LAST } ]} [, ...] ]
[ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
[ into_option ];
```

The specified subquery source **from_item** is as follows:

```
{[ ONLY ] {table_name | view_name} [ * ] [ partition_clause ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
|( select ) [ AS ] alias [ ( column_alias [, ...] ) ]
|with_query_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
|from_item [ NATURAL ] join_type from_item [ ON join_condition | USING ( join_column [, ...] ) ]}
```

If a subquery contains only one table, data is updated into the table. If a subquery contains multiple tables or has nested relationships, check whether a key-preserved table exists to determine whether data can be updated. For details about key-preserved tables and **WITH CHECK OPTION**, see **CREATE VIEW**.

- **view_name**

  Specifies the target view to be updated.

  NOTE

  The restrictions on updating views and subqueries are as follows:

  - The UPDATE operation can be performed only on columns that directly reference user columns in the base table.
  - A subquery or view must contain at least one updatable column. For details about updatable columns, see **CREATE VIEW**.
  - Views and subqueries that contain the DISTINCT, GROUP BY, HAVING, LIMIT, or OFFSET clause at the top layer are not supported.
  - Views and subqueries that contain set operations (UNION, INTERSECT, EXCEPT, and MINUS) at the top layer are not supported.
  - Views and subqueries whose target lists contain aggregate functions, window functions, or return set functions (such as array_agg, json_agg, and generate_series) are not supported.
  - Views with BEFORE or AFTER triggers but without INSTEAD OF triggers or INSTEAD rules are not supported.
  - Table types supported in views and subqueries include ordinary tables, temporary tables, global temporary tables, partitioned tables, level-2 partitioned tables, Ustore tables, and Astore tables.
  - Only one base table can be updated at a time in a multi-table join view or join subquery.
  - Join views or subqueries can update only key-preserved tables. If **CHECK OPTION** is specified, join columns cannot be updated. For details about the key-preserved table, see **CREATE VIEW**.
  - System views cannot be updated.

- **partition_clause**

  Updates a specified partition.

  PARTITION { ({ partition_name | subpartition_name } [, ...]) | FOR (partition_value [, ...]) } |

  SUBPARTITION { (subpartition_name) | FOR (subpartition_value [, ...]) }

  For details about the keywords, see **SELECT**.

For details, see **CREATE TABLE SUBPARTITION**.

NOTE

If PARTITION specifies multiple partition names, level-1 and level-2 partition names can coexist and can be the same. The union set of the partition range is used.

- **alias**

  Specifies a substitute name for the target table.

  Value range: a string. It must comply with the **naming convention**.

- **table_list**

  Specifies an expression list of a table. It is similar to **from_list**, but can declare both the target table and associated table. It is used only in the syntax for updating multiple tables. The items of **table_list** can be subqueries.

- **column_name**

  Specifies the name of the column to be modified.

  You can refer to this column by specifying the target table alias and the column name. Example: **UPDATE foo AS f SET f.col_name = 'namecol'**

  Value range: an existing column

- **expression**

  Specifies a value assigned to a column or an expression that assigns the value.

- **DEFAULT**

  Specifies the default value of a column.

  The value is **NULL** if no specified default value has been assigned to it.

- **sub_query**

  Specifies a subquery.

  This statement can be executed to update a table with information for other tables in the same database. For details about clauses in the SELECT statement, see **SELECT**.

  When a single column is updated, the ORDER BY and LIMIT clauses can be used. When multiple columns are updated, the ORDER BY and LIMIT clauses cannot be used.

- **from_list**

  Specifies a list of table expressions, allowing columns from other tables to appear in the WHERE condition and the update expressions. This is similar to the list of tables that can be specified in the FROM clause of a SELECT statement.

NOTICE

Note that the target table cannot appear in **from_list**, unless you intend a self-join (in which case it must appear with an alias in **from_list**).

- **condition**

  Specifies an expression that returns a value of type Boolean. Only rows for which this expression returns **true** are updated. You are advised not to use numeric types such as int for **condition**, because such types can be implicitly

converted to Boolean values (non-zero values are implicitly converted to **true** and **0** is implicitly converted to **false**), which may cause unexpected results.

- **WHERE CURRENT OF cursor_name**

  When the cursor points to a row in a table, you can use this syntax to update the row.

  cursor_name: specifies the name of a cursor.

  NOTICE

  - This syntax is not supported by the database in the B compatibility mode.
  - This syntax supports only ordinary tables instead of partitioned tables.
  - This syntax can be used only in stored procedures.
  - This syntax cannot be used together with other WHERE conditions.
  - Multi-table update is not supported.
  - This syntax cannot be used together with WITH, USING, ORDER BY, or FROM.
  - The SELECT statement corresponding to the CURSOR must be declared as FOR UPDATE.
  - The SELECT statement corresponding to the CURSOR supports only a single table. It does not support LIMIT/OFFSET, subqueries, or sublinks.
  - The CURSOR declared as FOR UPDATE in a stored procedure cannot be used after being committed or rolled back.
  - If the row to which the cursor points does not exist, an error is reported (only when UPDATE is used instead of DELETE) in the A compatibility mode, indicating that the specified row does not exist. In other compatibility modes, no error is reported.

- **ORDER BY**

  For details about the keywords, see **SELECT**.

- **LIMIT**

  For details about the keywords, see **SELECT**.

- **RETURNING output_expression**

  Specifies an expression to be computed and returned by the **UPDATE** statement after each row is updated.

  Value range: The expression can use any column names of the table named by **table_name** or tables listed in **FROM**. Write * to return all columns.

- **output_name**

  Specifies a name to use for a returned column.

Examples

- Modify all data in the table.

```
-- Create the tbl_test1 table and insert data into the table.
gaussdb=# CREATE TABLE tbl_test1(id int, info varchar(10));
gaussdb=# INSERT INTO tbl_test1 VALUES (1, 'A'), (2, 'B');

-- Query.
```



```
gaussdb=# SELECT * FROM tbl_test1;
 id | info
----+------
  1 | A
  2 | B
(2 rows)

-- Modify the info column of all data in the tbl_test1 table.
gaussdb=# UPDATE tbl_test1 SET info = 'aa';

-- Query the tbl_test1 table.
gaussdb=# SELECT * FROM tbl_test1;
 id | info
----+------
  1 | aa
  2 | aa
(2 rows)
```

- Modify some data in the table.

```
-- Modify the data whose ID is 2 in the tbl_test1 table.
gaussdb=# UPDATE tbl_test1 SET info = 'bb' WHERE id = 2;

-- Query the tbl_test1 table.
gaussdb=# SELECT * FROM tbl_test1;
 id | info
----+------
  1 | aa
  2 | bb
(2 rows)
```

- Modify the data and return the modified data.

```
-- Modify the data whose ID is 1 in the tbl_test1 table and specify the info column to be returned.
gaussdb=# UPDATE tbl_test1 SET info = 'ABC' WHERE id = 1 RETURNING info;
 info
------
 ABC
(1 row)

UPDATE 1

-- Delete the tbl_test1 table.
gaussdb=# DROP TABLE tbl_test1;
```

- Use a subquery to insert new data based on the existing data when modifying data.

```
-- Create a table.
gaussdb=# CREATE TABLE test_grade (
    sid int,                -- Student ID
    name varchar(50),       -- Name
    score char,             -- Score
    examtime date,          -- Exam time
    last_exam boolean       -- The last exam or not
);

-- Insert data.
gaussdb=# INSERT INTO test_grade VALUES (1,'Scott','A','2008-07-08',1),(2,'Ben','D','2008-07-08',1),
(3,'Jack','D','2008-07-08',1);

-- Query.
gaussdb=# SELECT * FROM test_grade;
 sid | name  | score |  examtime  | last_exam
-----+-------+-------+------------+-----------
   3 | Jack  | D     | 2008-07-08 | t
   1 | Scott | A     | 2008-07-08 | t
   2 | Ben   | D     | 2008-07-08 | t
(3 rows)

-- On August 25, 2008, Ben took a make-up exam and the score is B. You need to change the value of
last_exam to No, and then insert the score of August 25, 2008.
gaussdb=# WITH old_exam AS ( UPDATE test_grade SET last_exam = 0 WHERE sid = 2 AND
```



```
examtime = '2008-07-08' RETURNING sid, name )
                INSERT INTO test_grade VALUES ( ( SELECT sid FROM old_exam ), ( SELECT name FROM
old_exam ), 'B', '2008-08-25', 1 );

-- Query.
gaussdb=# SELECT * FROM test_grade;
 sid | name  | score |  examtime  | last_exam
-----+-------+-------+------------+-----------
   3 | Jack  | D     | 2008-07-08 | t
   1 | Scott | A     | 2008-07-08 | t
   2 | Ben   | D     | 2008-07-08 | f
   2 | Ben   | B     | 2008-08-25 | t
(4 rows)

-- Delete.
gaussdb=# DROP TABLE test_grade;
```

- **Updating a view or subquery**

  Example 1: Update a subquery.

```
-- Create a schema.
gaussdb=# CREATE SCHEMA upd_subqry;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA = 'upd_subqry';
SET

-- Create tables and insert data into the tables.
gaussdb=# CREATE TABLE t1 (x1 int, y1 int);
CREATE TABLE
gaussdb=# CREATE TABLE t2 (x2 int PRIMARY KEY, y2 int);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "t2_pkey" for table "t2"
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES (1, 1), (2, 2), (3, 3), (5, 5);
INSERT 0 4
gaussdb=# INSERT INTO t2 VALUES (1, 1), (2, 2), (3, 3), (5, 5);
INSERT 0 4

-- Update t1 through a subquery.
gaussdb=# UPDATE (SELECT * FROM t1) SET y1 = 13 where y1 = 3;
UPDATE 1
gaussdb=# UPDATE (SELECT * FROM t1 WHERE y1 < 2) SET y1 = 12 WHERE y1 = 2;
UPDATE 0

-- Insert a subquery with READ ONLY specified.
gaussdb=# UPDATE (SELECT * FROM t1 WITH READ ONLY) SET y1 = 1 WHERE y1 = 11;
ERROR:  cannot perform a DML operation on a read-only subquery.

-- Insert a multi-table join subquery.
gaussdb=# UPDATE (SELECT * FROM t1, t2 WHERE x1 = x2) SET y1 = 11 WHERE y2 = 1;
UPDATE 1

-- Insert a multi-table join subquery with CHECK OPTION specified. The join columns x1 and x2
cannot be updated.
gaussdb=# UPDATE (SELECT * FROM t1, t2 WHERE x1 = x2 WITH CHECK OPTION) SET y1 = 1 WHERE
y2 = 1;
UPDATE 1
gaussdb=# UPDATE (SELECT * FROM t1, t2 WHERE x1 = x2 WITH CHECK OPTION) SET x1 = 6 WHERE
y2 = 5;
ERROR:  virtual column not allowed here

-- Delete a schema.
gaussdb=# RESET CURRENT_SCHEMA;
RESET
gaussdb=# DROP SCHEMA upd_subqry CASCADE;
NOTICE:  drop cascades to 2 other objects
DETAIL:  drop cascades to table upd_subqry.t1
drop cascades to table upd_subqry.t2
DROP SCHEMA
```

  Example 2: Update a view.


```
-- Create a schema.
gaussdb=# CREATE SCHEMA upd_view;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA = 'upd_view';
SET

-- Create tables and insert data into the tables.
gaussdb=# CREATE TABLE t1 (x1 int, y1 int);
CREATE TABLE
gaussdb=# CREATE TABLE t2 (x2 int PRIMARY KEY, y2 int);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "t2_pkey" for table "t2"
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES (1, 1), (2, 2), (3, 3), (5, 5);
INSERT 0 4
gaussdb=# INSERT INTO t2 VALUES (1, 1), (2, 2), (3, 3), (5, 5);
INSERT 0 4

-- Create a single table view.
gaussdb=# CREATE VIEW v_upd1 AS SELECT * FROM t1;
CREATE VIEW
gaussdb=# CREATE VIEW v_upd_read AS SELECT * FROM t1 WITH READ ONLY;
CREATE VIEW

-- Update t1 through a view.
gaussdb=# UPDATE v_upd1 SET y1 = 13 where y1 = 3;
UPDATE 1
gaussdb=# UPDATE v_upd_read SET y1 = 1 WHERE y1 = 11;
ERROR:  cannot perform a DML operation on a read-only subquery.

-- Create a multi-table view.
gaussdb=# CREATE VIEW vv_upd AS SELECT * FROM t1, t2 WHERE x1 = x2;
CREATE VIEW
gaussdb=# CREATE VIEW vv_upd_wco AS SELECT * FROM t1, t2 WHERE x1 = x2 WITH CHECK OPTION;
CREATE VIEW

-- Update t1 through the join view.
gaussdb=# UPDATE vv_upd SET y1 = 1 WHERE y2 = 1;
UPDATE 1
gaussdb=# UPDATE vv_upd_wco SET x1 = 6 WHERE y2 = 5;
ERROR:  virtual column not allowed here

-- Delete a schema.
gaussdb=# RESET CURRENT_SCHEMA;
RESET
gaussdb=# DROP SCHEMA upd_view CASCADE;
NOTICE:  drop cascades to 6 other objects
DETAIL:  drop cascades to table upd_view.t1
drop cascades to table upd_view.t2
drop cascades to view upd_view.v_upd1
drop cascades to view upd_view.v_upd_read
drop cascades to view upd_view.vv_upd
drop cascades to view upd_view.vv_upd_wco
DROP SCHEMA
```

7.12.21 V

7.12.21.1 VACUUM

Description

VACUUM recycles storage space occupied by tables or B-Tree indexes. In normal database operation, rows that have been deleted are not physically removed from their table; they remain present until a VACUUM is done. Therefore, it is necessary to do VACUUM periodically, especially on frequently-updated tables.

Precautions

- If no table is specified, VACUUM processes the tables on which the user has the corresponding permission in the current database. With a parameter, VACUUM processes only that table.
- To perform VACUUM operation on a table, you must be a table owner or a user granted the VACUUM permission on the table. When the separation of duties is disabled, system administrators have this permission by default. However, database owners are allowed to VACUUM all tables in their databases, except shared catalogs. (The restriction for shared catalogs means that a true database-wide VACUUM can only be executed by the system administrator). VACUUM skips over any tables that the calling user does not have the permission to vacuum.
- VACUUM cannot be executed inside a transaction block.
- It is recommended that active production databases be vacuumed frequently (at least nightly), in order to remove dead rows. After adding or deleting a large number of rows, it might be a good idea to run **VACUUM ANALYZE** for the affected table. This will update the system catalogs with the results of all recent changes, and allow the query planner to make better choices in planning queries.
- FULL is recommended only in special scenarios. For example, you wish to physically narrow the table to decrease the occupied disk space after deleting most rows of a table. VACUUM FULL usually shrinks a table more than VACUUM does. The **FULL** option does not clear indexes. You are advised to periodically use the **REINDEX** statement. If the physical space usage does not decrease after you run the statement, check whether there are other active transactions (that have started before you delete data transactions and not ended before you run **VACUUM FULL**). If there are such transactions, run this statement again when the transactions quit.
- VACUUM FULL returns the free space in the table to the tablespace by rebuilding the table. During the rebuilding, extra storage space equivalent to the valid data size in the table is required. For a non-segment-paged table, after VACUUM FULL is executed, the physical files occupied by the original table are deleted, and the physical file space occupied by the original table is returned to the operating system. For a segment-paged table, after VACUUM FULL is executed, the physical space occupied by the original table is returned to the segment-paged data file instead of the operating system.
- VACUUM causes a substantial increase in I/O traffic, which might cause poor performance for other active sessions. Therefore, it is sometimes advisable to use the cost-based VACUUM delay feature.
- When **VERBOSE** is specified, VACUUM prints progress messages to indicate which table is currently being processed. Various statistics about the tables are printed as well.
- When the option list is surrounded by parentheses, the options can be written in any order. If there are no brackets, the options must be given in the order displayed in the syntax.
- VACUUM and VACUUM FULL clear deleted tuples after the delay specified by **vacuum_defer_cleanup_age**.
- VACUUM ANALYZE executes a VACUUM operation and then an ANALYZE operation for each selected table. This is a handy combination form for routine maintenance scripts.

- Plain VACUUM (without FULL) simply recycles space and makes it available for reuse. This form of statement can operate in parallel with normal reading and writing of the table, as an exclusive lock is not obtained. VACUUM FULL executes wider processing, including moving rows across blocks to compress tables so they occupy the minimum number of disk blocks. This form is much slower and requires an exclusive lock on each table while it is being processed.
- A deadlock may occur when multiple VACUUM FULL statements are executed simultaneously.
- If the **xc_maintenance_mode** parameter is not enabled, the VACUUM FULL operation will skip all system catalogs.
- If you run **VACUUM FULL** immediately after running **DELETE**, the space will not be recycled. After running **DELETE**, execute 1000 non-SELECT transactions, or wait for 1s and then execute one transaction. Then, run **VACUUM FULL** to the space.
- During VACUUM FULL, an exclusive lock is added to the table. Therefore, you are advised not to run **VACUUM FULL** during peak hours because it may cause long waiting time or service interruption.
- For Ustore, the behavior of manual VACUUM is the same as that in Astore. Locks are obtained to clear heap tables and indexes. In Ustore, AUTOVACUUM only clears GPIs of partitioned tables, updates FSMs of heap tables, and recycles index pages.
- When **VACUUM FULL** is executed, partitions are traversed for clearance and GPIs are rebuilt after partition clearance. Therefore, if there are a large number of partitions, you are advised to delete GPIs first and rebuild indexes after **VACUUM FULL** is executed. In this way, the execution time of **VACUUM FULL** is reduced.

Syntax

- Recycle space and update statistics information, without requirements for keyword orders.

```
VACUUM [ ( { FULL | FREEZE | VERBOSE | {ANALYZE | ANALYSE }} [,...] ) ]
    [ table_name [ (column_name [, ...] ) ] [ PARTITION ( partition_name ) | SUBPARTITION
( subpartition_name ) ] ];
```

- Recycle space, without updating statistics information.

```
VACUUM [ FULL [COMPACT] ] [ FREEZE ] [ VERBOSE ] [ table_name
[ PARTITION ( partition_name )  | SUBPARTITION ( subpartition_name ) ] ];
```

- Recycle space and update statistics information, and require keywords in order.

```
VACUUM [ FULL ] [ FREEZE ] [ VERBOSE ] { ANALYZE | ANALYSE } [ VERBOSE ]
    [ table_name [ (column_name [, ...] ) ] ] [ PARTITION ( partition_name ) ];
```

- Recycle the space of a specified bucket and update the **bucketxid** column in the pg_hashbucket system catalog. Statistics are not updated. This function is not supported in the current version.

```
VACUUM FREEZE BUCKETS (bucketid [, ...]);
```

Parameters

- **FULL**

  Selects "FULL" vacuum, which can recycle more space, but takes much longer and exclusively locks the table.

> NOTE
>
> Using **FULL** will cause statistics missing. To collect statistics, add the keyword **ANALYZE** to **VACUUM FULL**.

- **FREEZE**

  Is equivalent to running **VACUUM** with the **vacuum_freeze_min_age** parameter set to **0**.

- **VERBOSE**

  Prints a detailed VACUUM activity report for each table.

- **ANALYZE | ANALYSE**

  Updates statistics used by the planner to determine the most efficient way to execute a query.

  > NOTE
  >
  > VACUUM is also triggered when **autovacuum** is set to **analyze** for a Ustore partitioned table.

- **table_name**

  Specifies the name (optionally schema-qualified) of a specific table to vacuum.

  Value range: name of a specific table to vacuum Defaults are all tables in the current database.

- **column_name**

  Specifies the name of the column to be analyzed. This parameter must be used together with ANALYZE.

  Value range: name of the column to be analyzed Defaults are all columns.

  > NOTE
  >
  > The mechanism of the VACUUM ANALYZE statement is to execute VACUUM and ANALYZE in sequence. Therefore, if **column_name** is incorrect, VACUUM may be successfully executed but ANALYZE may fail to be executed. For a partitioned table, ANALYZE may fail to be executed after VACUUM is successfully executed on a partition.

- **PARTITION**

  COMPACT and **PARTITION** cannot be used at the same time.

- **partition_name**

  Specifies the level-1 partition name of the table to be cleared. If it is left empty, all level-1 partitions are cleared.

- **subpartition_name**

  Specifies the level-2 partition name of the table to be cleared. If it is left empty, all level-2 partitions are cleared.

Examples

- VACUUM

```
-- Create the table tbl_test and insert data into the tables.
CREATE TABLE tbl_test(c1 int);
INSERT INTO tbl_test VALUES (1);

-- View the data and the CTID of the data.
SELECT ctid,* FROM tbl_test;
```



```
 ctid  | c1
-------+----
 (0,1) |  1
(1 row)

-- Delete the data record.
DELETE FROM tbl_test;

-- Insert a data record again. It is found that a new ctid is used.
INSERT INTO tbl_test VALUES (2);
SELECT ctid,* FROM tbl_test;
 ctid  | c1
-------+----
 (0,2) |  2
(1 row)

-- After the VACUUM statement is executed, the old space is reused when data is inserted.
VACUUM ANALYZE tbl_test;
INSERT INTO tbl_test VALUES (3);
SELECT ctid,* FROM tbl_test;
 ctid  | c1
-------+----
 (0,1) |  3
 (0,2) |  2
(2 rows)

-- Delete the table.
DROP TABLE tbl_test;
```

- VACUUM FULL

```
-- Create a table.
CREATE TABLE tbl_test2(c1 int);

-- Insert 100,000 data records and check the table size.
INSERT INTO tbl_test2 VALUES (generate_series(1,100000));
SELECT 'tbl_test2' AS tablename, pg_size_pretty(pg_relation_size('tbl_test2')) AS size;
 tablename |  size
-----------+---------
 tbl_test2 | 3048 kB
(1 row)

-- Delete data and check the table size.
DELETE FROM tbl_test2;
SELECT 'tbl_test2' AS tablename, pg_size_pretty(pg_relation_size('tbl_test2')) AS size;
 tablename |  size
-----------+---------
 tbl_test2 | 3048 kB
(1 row)

-- Use VACUUM FULL to reclaim space and check the table size.
VACUUM FULL ANALYZE tbl_test2;
SELECT 'tbl_test2' AS tablename, pg_size_pretty(pg_relation_size('tbl_test2')) AS size;
 tablename |  size
-----------+---------
 tbl_test2 | 0 bytes
(1 row)


-- Delete.
DROP TABLE tbl_test2;
```

Suggestions

- VACUUM
  - VACUUM cannot be executed inside a transaction block.
  - It is recommended that active production databases be vacuumed frequently (at least nightly), in order to remove dead rows. It is strongly

recommended that you run **VACUUM ANALYZE** after adding or deleting a large number of records.

- **FULL** is recommended only in special scenarios. For example, you wish to physically narrow the table to decrease the occupied disk space after deleting most rows of a table.

7.12.21.2 VALUES

Function

Computes a row or a set of rows based on given values. It is most commonly used to generate a constant table within a large statement.

Precautions

- **VALUES** lists with large numbers of rows should be avoided, as you might encounter out-of-memory failures or poor performance. **VALUES** appearing within **INSERT** is a special case, because the desired column types are known from the **INSERT**'s target table, and need not be inferred by scanning the **VALUES** list. In this case, a large number of result rows can be processed.
- If more than one row is specified, all the rows must have the same number of elements.

Syntax

```
VALUES {( expression [, ...] )} [, ...]
    [ ORDER BY { sort_expression [ ASC | DESC | USING operator ] } [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ];
```

Parameter Description

- **expression**

  Specifies a constant or expression to compute and insert at the indicated place in the resulting table or set of rows.

  In a **VALUES** list appearing at the top level of an **INSERT**, an expression can be replaced by **DEFAULT** to indicate that the destination column's default value should be inserted. **DEFAULT** cannot be used when **VALUES** appears in other contexts.

- **sort_expression**

  Specifies an expression or integer constant indicating how to sort the result rows.

- **ASC**

  Specifies an ascending sort order.

- **DESC**

  Specifies a descending sort order.

- **operator**

  Specifies a sorting operator.

- **count**

  Specifies the maximum number of rows to return.

- **OFFSET start [ROW | ROWS]**

  Specifies the maximum number of returned rows, whereas **start** specifies the number of rows to skip before starting to return rows.

- **FETCH { FIRST | NEXT } [count] { ROW | ROWS } ONLY**

  The **FETCH** clause restricts the total number of rows starting from the first row of the return query result, and the default value of **count** is **1**.

Examples

See **Examples** in **INSERT**.

7.13 Appendix

7.13.1 Extended Functions

The following table lists the extended functions supported by GaussDB. These functions are for reference only.

Category	Name	Description
Access privilege inquiry function	has_sequence_privilege(user, sequence, privilege)	Queries whether a specified user has privilege for sequences.
	has_sequence_privilege(sequence, privilege)	Queries whether the current user has privilege for sequence.
Trigger function	pg_get_triggerdef(oid)	Gets **CREATE** [**CONSTRAINT**] **TRIGGER** command for triggers.
	pg_get_triggerdef(oid, boolean)	Gets **CREATE** [**CONSTRAINT**] **TRIGGER** command for triggers.

7.13.2 Extended Syntax

GaussDB provides the following extended syntax:

Table 7-227 Extended SQL syntax

Category	Keywords	Description
Creating a table	column_constraint: **REFERENCES reftable [(refcolumn)] [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE][ON DELETE action] [ON UPDATE action]**	You can run **REFERENCES reftable[(*refcolumn*)] [MATCH FULL |MATCH PARTIAL | MATCH SIMPLE] [ON DELETE action] [ON UPDATE action]** to create foreign key constraints for tables.
Loading a module	**CREATE EXTENSION**	Loads a new module to the current database. This feature is for internal use only. You are advised not to use it.
	DROP EXTENSION	Deletes the loaded module. This feature is for internal use only. You are advised not to use it.
Aggregate functions	**CREATE AGGREGATE**	Defines a new aggregate function.
	ALTER AGGREGATE	Modifies the definition of an aggregate function.
	DROP AGGREGATE	Drops an existing function.

7.13.3 Dollar-Quoted String Constants

If a string sequence contains a single quotation mark ('), double the single quotation mark (') to two single quotation marks (''). Otherwise, the SQL statement may fail to be executed.

If a string contains many single quotation marks (') or backslashes (\), the string may be difficult to understand and error-prone because all the single quotation marks (') are doubled.

To make the query more readable in such situations, dollar quoting is provided to write string constants. A dollar-quoted string constant consists of a dollar sign ($), a "tag" of zero or more characters, another dollar sign, an arbitrary sequence of characters that make up the string content, a dollar sign, another tag the same as the previous one, and a dollar sign.

```
gaussdb=# SELECT $$it's an example$$;
   ?column?
-----------------
 it's an example
(1 row)
```


7.13.4 DATABASE LINK

Description

In the local database, a database link is used to connect to and access the remote database.

A database link can be public or private. A private database link can be accessed only by the creator, while a public database link can be accessed by all users.

All created database link information is stored in the system view gs_db_links of the local database.

Precautions

- The database link feature can be used only in A-compatible versions.
- The remote database connected to database link supports only 503.1.0 and later versions.
- Ensure that the values of the compatibility parameter **DBCOMPATIBILITY** and GUC parameters **behavior_compat_options**, **a_format_dev_version**, and **a_format_version** of the local and remote databases are the same.
- When a session is enabled for a database link connection, the following GUC parameters are set:

  ```
  set search_path=pg_catalog, '$user', 'public';
  set datestyle=ISO;
  set intervalstyle=postgres;
  set extra_float_digits=3;
  ```

  Other parameters are set at the remote end. If the remote parameters are different from the local parameters, the data display formats may be different. Therefore, ensure that the remote parameters are the same as the local parameters.
- Preparations: Use gs_guc to add a whitelist to the **gs_hba.conf** file to allow client connections.

  ```
  Example: gs_guc reload -I all -N all -Z datanode -h "host all all 192.168.11.11/32 sha256"
  ```

  For details about the parameters, see the description of gs_guc client authentication policy settings.
- The permission to create a database link needs to be granted using the GRANT syntax. By default, a new user does not have the permission, but the system administrator has the permission. For details, see GRANT description.
- When a database link is used to perform operations on a remote table, a schema corresponding to the remote table is created locally. If the metadata of the table does not exist locally, the metadata is written to the local system catalog. In this case, a level-7 lock is used to ensure write consistency until the transaction ends. When a database link is deleted, the corresponding metadata is also deleted.
- When DATABASE LINK is used, locally created tables are used only to store metadata of remote tables. The table structure cannot be queried using the \d or pg_get_tabledef function.
- If a long transaction uses the database link to operate a remote object for the first time, the lock is held until the transaction ends. Other transactions that use the dblink for the first time are blocked. To avoid this problem, run a quick statement, for example, "select * from t1@dblink where 1=2;", to query

the remote object to be used and flush its metadata to disks. In addition, similar problem also occurs when the structure of the remote table changes and the stored metadata is updated locally.

- If the local and remote character sets are different, an error indicating that the conversion fails may be reported. The error information is that the remote end returns an error. If the character encoding of the local database is GB18030_2022, the character encoding sent to the remote database is converted to GB18030. Therefore, if the character set of the local database is GB18030_2022, the character set of the remote database can only be GB18030 or GB18030_2022.
- When a schema corresponding to the remote end is created locally, "USERNAME (available only for private database link) #remote schema@DBLINK" is used as the schema name. The maximum length of the schema name is 63 characters.

NOTICE

When the permission to create a database link is granted to a user, the user can remotely access a database by using the IP address of the remote database. Exercise caution when granting this permission to users.

Syntax

- Create a database link.

```
CREATE [ PUBLIC ] DATABASE LINK dblink
  [ CONNECT TO { CURRENT_USER | user IDENTIFIED BY password } ] [ USING ( option 'value' [...]) ];
```

- Modify the database link information.

```
ALTER [ PUBLIC ] DATABASE LINK dblink
  { CONNECT TO user IDENTIFIED BY password };
```

- Delete a specified database link.

```
DROP [ PUBLIC ] DATABASE LINK dblink ;
```


NOTE

- **PUBLIC**: creates a public database link visible to all users. If this clause is omitted, the database link is private and available only to the current user. The data that can be accessed on the remote database depends on the identity used by the database link during connection.
- If **CONNECT TO user IDENTIFIED BY password** is specified, the database link makes a connection as a user with specified password.
- If **CONNECT TO CURRENT_USER** is specified, the database link connects to the remote database as the local initial user.
- If the preceding two clauses are omitted, the database link connects to the remote database as a local initial user.
- **dblink**: indicates the name of the database link to be created.
- **user**: indicates the username used by the created database link.
- **password**: indicates the password of the username.
- **USING (option 'value' [, ...])**

  Specifies parameters such as the IP address, port number, and remote database name of the database to be connected. The supported options are as follows:

  - **host**: specifies the IP addresses to be connected. IPv6 addresses are not supported. Multiple IP addresses can be specified using character strings separated by commas (,). Currently, encrypted databases, SSL settings, and certificate authentication are not supported. If no IP address is specified, this parameter is left empty by default.
  - **port**: specifies the port number for connection. If this parameter is not specified, the default value **5432** is used.
  - **dbname**: specifies the name of the database to be connected. If this parameter is not specified, the username used for connecting to the remote end is used by default.
  - **fetch_size**: specifies the amount of data obtained from the remote end each time. The value of **fetch_size** ranges from 0 to 2147483647. The default value is **100**.

  Notes:

  - You can write only part of the preceding options in the brackets after USING.
  - If the keyword USING is not written, the content in the brackets is not written as well.
  - When a database link is created, the system does not check whether the connection is successful. If related keywords are missing, an error may be reported.

- Perform the SELECT operation through a database link.

  NOTE

  The syntax for accessing a remote database object by using a created database link is basically the same as that for accessing a local object. The difference is that **@dblink** is added to the end of the remote object descriptor. For details about restrictions on SQL statements, see **Table 7-228**.

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
 SELECT [/*+ plan_hint */] [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
{ * | {expression [ [ AS ] output_name ]} [, ...] }
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ [ START WITH condition ] CONNECT BY [NOCYCLE] condition [ ORDER SIBLINGS BY expression ] ]
[ GROUP BY grouping_element [, ...] ]
[ HAVING condition [, ...] ]
[ { UNION | INTERSECT | EXCEPT | MINUS } [ ALL | DISTINCT ] select ]
[ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS
```



```
{ FIRST | LAST } ]} [, ...] ]
 [ LIMIT { [offset,] count | ALL } ]
 [ OFFSET start [ ROW | ROWS ] ]
 [ {FOR { UPDATE | SHARE} [ OF table_name [, ...] ] } [...] ];
 {[ ONLY ] table_name [ * ] [ partition_clause ] @ dblink  [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
  |( select ) [ AS ] alias [ ( column_alias [, ...] ) ]
  |with_query_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
  |[function_name] ( [ argument [, ...] ] ) [ AS ] alias [ ( column_alias [, ...] | column_definition [, ...] ) ]
  |[function_name] ( [ argument [, ...] ] ) AS ( column_definition [, ...] )
  |from_item [ NATURAL ] join_type from_item [ ON join_condition | USING ( join_column [, ...] ) ]};
```

- Perform the INSERT operation through a database link.

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
  INSERT [/*+ plan_hint */] INTO table_name [ partition_clause ] @ dblink [ ( column_name [, ...] ) ]
     { DEFAULT VALUES
     | VALUES {( { expression | DEFAULT } [, ...] ) }[, ...]
     | query }
     [ RETURNING { {output_expression [ [ AS ] output_name ] }[, ...]} ];
```

- Perform the UPDATE operation through a database link.

```
UPDATE [/*+ plan_hint */] [ ONLY ] table_name [ partition_clause ] @ dblink  [ [ AS ] alias ]
 SET {column_name = { expression | DEFAULT }
    |( column_name [, ...] ) = {( { expression | DEFAULT } [, ...] ) |sub_query }}[, ...]
    [ FROM from_list ] [ WHERE condition ]
    [ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS
{ FIRST | LAST } ]} [, ...] ]
    [ LIMIT { [offset,] count | ALL } ]
    [ RETURNING { {output_expression [ [ AS ] output_name ]} [, ...] }];
 where sub_query can be:
 SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
 { * | {expression [ [ AS ] output_name ]} [, ...] }
 [ FROM from_item [, ...] ]
 [ WHERE condition ]
 [ GEOUP BY grouping_element [, ...] ]
 [ HAVING condition [, ...] ];
```

- Perform the DELETE operation through a database link.

```
 [ WITH [ RECURSIVE ] with_query [, ...] ]
 DELETE [/*+ plan_hint */] FROM [ ONLY ] table_name [ partition_clause ] @ dblink  [ [ AS ] alias ]
    [ USING using_list ]
    [ WHERE condition]
    [ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause ] [ NULLS
{ FIRST | LAST } ]} [, ...] ]
    [ LIMIT { [offset,] count | ALL } ]
    [ RETURNING { * | { output_expr [ [ AS ] output_name ] } [, ...] } ];
```

- Perform the LOCK TABLE operation through a database link.

```
 LOCK [ TABLE ] {[ ONLY ] name @ dblink  [, ...]}

    [ IN {ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE | SHARE |
SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE} MODE ]
    [ NOWAIT ];
```

- Call a stored procedure or function of the remote database.

```
CALL | SELECT [ schema. ] { func_name@dblink | procedure_name@dblink } ( param_expr );
```


CAUTION

- When a database link calls a remote function or stored procedure, the user-defined types, **out** output parameters, aggregate functions, window functions, and return set functions are not supported. **SELECT * FROM func@dblink()** cannot be used to call the function or stored procedure.
- When a database link calls a remote function or stored procedure, functions under PUBLIC are called by default if no schema is specified.
- When the database link calls a remote function or stored procedure, **param_expr** does not support the use of ":=" or "=>" to separate parameter names and parameter values.

NOTE

- The meanings of the parameters irrelevant to database link in the preceding SQL statements are the same as those in the original SQL statements.
- When specifying a column name, you can add "@dblink" to the end of the column name to specify the column of the table to which the corresponding database link points.

Restrictions

- **Transaction**

  When a database link is used, the relationship between local and remote transactions is as follows:

  a. A local transaction synchronously controls the commit/rollback status of a remote transaction.

  b. The relationship between isolation levels is as follows.

Local Isolation Level	Remote Isolation Level
Read Uncommitted	Repeatable Read
Read Committed	Repeatable Read
Repeatable Read	Repeatable Read
Serializable	Serializable

NOTICE

If you commit a local transaction, a transaction commit request is sent to the remote end. If the local transaction fails to be committed due to an exception (such as a connection exception or a local cluster instance exception) after the remote transaction is successfully committed, the committed remote transaction cannot be withdrawn. As a result, the local transaction may be inconsistent with the remote transaction.

- **Permissions of local users to use database links**

a. If the keyword public is used, the link is a public database link and can be used by all users or schemas.

b. If the keyword public is not used, the link is a private database link and can be used only by the current user or schema. (User **sys** cannot use database links across schemas.)

- **Permission to access remote database objects through database links**

  The permissions to access remote database objects are the same as those of the remote connection user bound to the database link.

- **Supported SQL statements**
  - For details about the statements supported by database links, see **Table 7-228**.
  - For details about the table types supported by database links, see **Table 7-229**.
- **Function call using a database link**
  - When a database link calls a remote function, the user-defined types, OUT/INOUT parameters, PACKAGE inner functions, aggregate functions, window functions, and return set functions are not supported.
  - When a database link calls a stored procedure or function of a remote database in the PLSQL_BODY, the user-defined types, OUT/INOUT parameters, PACKAGE inner functions, overloaded functions, aggregate functions, window functions, and return set functions are not supported.

NOTICE

- You can use the following syntax to call stored procedures or functions of a remote database in the PLSQL_BODY: [CALL | SELECT] [schema.] { func_name@dblink | procedure_name@dblink } (param_expr)
- You can use the following syntax to call parameterless stored procedures or functions of a remote database in the PLSQL_BODY: [CALL | SELECT] [schema.] { func_name@dblink | procedure_name@dblink } ( ).

- **Synonyms**
  - The database link name cannot be created as a synonym.
  - Synonyms that point to a database link object in a remote database cannot be called through database link. Example:
    - Step 1: Create table **TABLE1** on **DB1**.
    - Step 2: Create **DBLINK1** on **DB2** for connecting to **DB1**. Create the synonym "CREATE SYNONYM T1 FOR TABLE1@DBLINK1".
    - Step 3: Create **DBLINK2** on **DB3** for connecting to **DB2**. Call the synonym T1 "SELECT * FROM T1@DBLINK2" on **DB2** through **DBLINK2**.
- **Table type constraints**
  - HASHBUCKET: The query or DML operation cannot be performed on the remote hash bucket table through database link.
  - SLICE: The query or DML operation cannot be performed on the remote slice table through database link.

  - Replication table: The query or DML operation cannot be performed on the remote replication table through database link.
  - TEMPORARY: The query or DML operation cannot be performed on the remote temporary table through database link.

- **Views**
  - Currently, you can create a view for a remote table of a database link. However, when the structure of the remote table changes, an exception may occur when you use the view. Example:
    - Step 1: Create table **TABLE1** on **DB1**.
    - Step 2: Create **DBLINK** on **DB2** for connecting to **DB1**. Create the view "CREATE VIEW V1 AS SELECT * FROM TABLE1@DBLINK".
    - Step 3: Delete a column from **TABLE1** on **DB1**. An error is reported when you query the view on **DB2**.
- **Other scenarios**
  - The database link table does not support triggers, including the scenarios where a database link is used in the function called by a trigger, the function called by a trigger is a database link function, and the trigger is defined on a database link.
  - The UPSERT and MERGE syntaxes are not supported.
  - The CURRENT CURSOR syntax is not supported.
  - Hidden columns in a table cannot be queried.
- **Dump and backup**

  Database objects related to database links cannot be dumped. The standby node cannot be called or connected by database links.
- **Predicate pushdown constraints**

  Only the data types, operators, and functions used in the WHERE clause are built in, and the used functions are of the IMMUTABLE type.
- **Aggregate function pushdown constraints**

  Only single tables that do not contain GROUP, ORDER BY, HAVING, and LIMIT clauses in the SELECT statement are supported. Window functions are not supported.
- **Hint pushdown**

  Only hints in scan mode can be pushed down based on the hint conditions of database link table objects. The syntax format is as follows:

  ```
  [no] tablescan|indexscan|indexonlyscan(table [index])
  ```

  The table name or table alias in a query block must be unique.

Table 7-228 Supported SQL statements

SQL Type	Operation Object	Supported Option	Execution Context
Creating a database link	DATABASE LINK	N/A	Common transaction block

SQL Type	Operation Object	Supported Option	Execution Context
Modifying a database link	DATABASE LINK	Only the username and password can be changed.	Common transaction block
Deleting a database link	DATABASE LINK	N/A	Common transaction block
SELECT statement	Ordinary table, common view, and complete-refresh materialized view	• WHERE clause • JOIN clause used between a database link table and an internal table • JOIN clause used between database link tables • Aggregate function • LIMIT clause • ORDER BY clause • GROUP BY and HAVING clauses • UNION clause • WITH clause • START WITH and CONNECT BY clauses • FOR UPDATE clause • ROWNUM	Common transaction block, stored procedure, function, advanced package, and logical view
INSERT statement	Ordinary table	• Inserting multiple values	Common transaction block, stored procedure, function, and advanced package
UPDATE statement	Ordinary table	• LIMIT clause • ORDER BY clause • WHERE clause	Common transaction block, stored procedure, function, and advanced package

SQL Type	Operation Object	Supported Option	Execution Context
DELETE statement	Ordinary table	• LIMIT clause • ORDER BY clause • WHERE clause	Common transaction block, stored procedure, function, and advanced package
LOCK TABLE statement	Ordinary table	• LOCKMODE clause • NOWAIT clause	Common transaction block

Table 7-229 Supported table types

Dimension	GaussDB Table Type		Database Link Support
TEMP option	Temporary table		Not supported
	Global temporary table		Supported
UNLOGGED option	Unlogged table		Supported
Storage features	Row store	Astore	Supported
		Ustore	Supported
	Partitioned table		Supported
	Level-2 partitioned table		Supported
Views	Remote view accessed by a database link		DQL statements are supported, but DML statements are not supported.
	Remote table associated with a local view through a database link		DQL statements are supported, but DML statements are not supported.

Examples

```
-- DDL statements
CREATE USER user2 WITH PASSWORD '********'; -- Create a common user.
GRANT CREATE PUBLIC DATABASE LINK TO user2; -- Grant the permission to create database links to a user.
GRANT DROP PUBLIC DATABASE LINK TO user2;  -- Grant the permission to drop database links to a user.
GRANT ALTER PUBLIC DATABASE LINK TO user2;  -- Grant the permission to modify database links to a user.
REVOKE CREATE PUBLIC DATABASE LINK FROM user2; -- Revoke the permission to create database links to a user.
REVOKE DROP PUBLIC DATABASE LINK FROM user2; -- Revoke the permission to drop database links to a user.
```



```
REVOKE ALTER PUBLIC DATABASE LINK FROM user2;   -- Revoke the permission to modify database links
to a user.
CREATE PUBLIC DATABASE LINK dblink CONNECT TO 'user1' IDENTIFIED BY '********' USING (HOST
'192.168.11.11', PORT '5432', DBNAME 'db1'); -- Create a database link object
ALTER PUBLIC DATABASE LINK dblink CONNECT TO 'user1' IDENTIFIED BY '********'; -- Modify database link
information.
DROP PUBLIC DATABASE LINK dblink; -- Delete a database link object.

-- Database link statements
-- Perform pre-operations.
CREATE USER user1 WITH SYSADMIN PASSWORD '********';
CREATE USER user2 WITH SYSADMIN PASSWORD '********';
CREATE DATABASE  db1;-- Remote database
CREATE DATABASE db2; -- Database for testing the database link.
\c db1 user1
-- Create an ordinary table.
db1=> CREATE TABLE remote_tb(f1 int, f2 text, f3 text[]);
db1=> INSERT INTO remote_tb VALUES (0,'a','{"a0","b0","c0"}');
db1=> INSERT INTO remote_tb VALUES (1,'bb','{"a1","b1","c1"}');
db1=> INSERT INTO remote_tb VALUES (2,'cc','{"a2","b2","c2"}');
-- Create a function.
CREATE OR REPLACE FUNCTION  f(a in int, b in int)
return int AS
    tmp int :=  a + b;
    BEGIN
        RETURN tmp;
    END;
/
CREATE FUNCTION
-- Create a synonym.
db1=> CREATE SYNONYM remote_sy FOR remote_tb;

\c db2 user2
db2=> CREATE TABLE local_tb(f1 int, f2 text, f3 text[]);
db2=> INSERT INTO local_tb VALUES (2,'c','{"a2","b2","c2"}');
db2=> CREATE PUBLIC DATABASE LINK dblink CONNECT TO 'user1' IDENTIFIED BY '********' USING (HOST
'192.168.11.11', PORT '5432', DBNAME 'db1'); -- Set host and port based on actual situation.
db2=> SELECT * FROM remote_tb@dblink; -- Query the remote table.
 f1 | f2 |     f3
----+----+------------
  1 | bb | {a1,b1,c1}
  2 | cc | {a2,b2,c2}
  0 | a  | {a0,b0,c0}
(3 rows)
db2=> INSERT INTO remote_tb@dblink VALUES (4,'d','{"a1","b2","c3"}'); -- Insert data into a remote table.
INSERT 0 1
db2=> UPDATE remote_tb@dblink SET f2 = 'aa' WHERE f1 = 0; -- Update the remote table.
UPDATE 1
db2=> DELETE remote_tb@dblink WHERE f1 = 1; -- Delete data from a remote table.
DELETE 1
db2=> SELECT * FROM remote_tb@dblink JOIN local_tb ON local_tb.f1 = remote_tb.f1@dblink; -- Join a
local table to a remote table.
 f1 | f2 |     f3     | f1 | f2 |     f3
----+----+------------+----+----+------------
  2 | cc | {a2,b2,c2} |  2 | c  | {a2,b2,c2}
(1 row)

db2=> SELECT count(*) FROM remote_tb@dblink;
 count
-------
     3
(1 row)
db2=>
db2=> SELECT f@dblink(1,2); -- Access the remote function.
 f
---
 3
(1 row)
```



```
CREATE OR REPLACE FUNCTION call_f(a in int, b in int) -- Access remote functions in PLSQL_BODY.
RETURN int AS
    tmp int;
    BEGIN
        tmp := f@dblink(a, b);
    RETURN tmp;
    END;
/
CREATE FUNCTION
db2=> SELECT call_f(1, 2);
 call_f
--------
      3
(1 row)
db2=> CREATE SYNONYM local_sy FOR remote_tb@dblink; -- Create a synonym for a database link object.
CREATE SYNONYM
db2=> SELECT * FROM local_sy;
 f1 | f2 |     f3
----+----+------------
  1 | bb  | {a1,b1,c1}
  2 | cc | {a2,b2,c2}
  0 | a  | {a0,b0,c0}
(3 rows)
db2=> SELECT * FROM remote_sy@dblink; -- Access the remote database synonym.
 f1 | f2 |     f3
----+----+------------
  1 | bb | {a1,b1,c1}

  2 | cc | {a2,b2,c2}
  0 | a  | {a0,b0,c0}
(3 rows)
db2=> EXPLAIN VERBOSE SELECT /*+ tablescan(remote_sy) */ * FROM remote_sy@dblink; -- Partial hint
pushdown supported by the database link.
                                QUERY PLAN
---------------------------------------------------------------------------------
 Foreign Scan on public.remote_tb remote_sy  (cost=100.00..100.03 rows=1 width=68)
   Output: f1, f2, f3
   Remote SQL: SELECT /*+  tablescan(remote_sy) */ f1, f2, f3 FROM public.remote_tb
(3 rows)

db2=> SELECT * FROM gs_database_link; -- View the database link system catalog.
db2=> START TRANSACTION;
START TRANSACTION
db2=> SELECT * FROM remote_sy@dblink;
 f1 | f2 |     f3
----+----+------------
  1 | bb | {a1,b1,c1}
  2 | cc | {a2,b2,c2}
  0 | a  | {a0,b0,c0}
(3 rows)

db2=> SELECT intransaction FROM gs_db_links; -- Check the database link system view.
 intransaction
---------------
 t
(1 row)
db2=> END;
COMMIT
db2=> ALTER PUBLIC DATABASE LINK dblink CONNECT TO 'user1' IDENTIFIED BY '********'; -- Modify
database link information.
db2=> DROP PUBLIC DATABASE LINK dblink; -- Delete a database link object.
```

Helpful Links

CREATE DATABASE LINK, **GS_DATABASE_LINK**, and **GS_DB_LINKS**

7.13.5 Row Expression Function Whitelist

Table 7-230 Expression function whitelist for data objects to add or modify ILM policies

func_oid_value	func_name
56	boollt
57	boolgt
60	booleq
61	chareq
63	int2eq
64	int2lt
65	int4eq
66	int4lt
67	texteq
70	charne
72	charle
73	chargt
74	charge
77	int4
78	char
84	boolne
111	numeric_fac
141	int4mul
144	int4ne
145	int2ne
146	int2gt
147	int4gt
148	int2le
149	int4le
150	int4ge
151	int2ge
152	int2mul

func_oid_value	func_name
153	int2div
154	int4div
155	int2mod
156	int4mod
157	textne
158	int24eq
159	int42eq
160	int24lt
161	int42lt
162	int24gt
163	int42gt
164	int24ne
165	int42ne
166	int24le
167	int42le
168	int24ge
169	int42ge
170	int24mul
171	int42mul
172	int24div
173	int42div
176	int2pl
177	int4pl
178	int24pl
179	int42pl
180	int2mi
181	int4mi
182	int24mi
183	int42mi
202	float4mul

func_oid_value	func_name
203	float4div
204	float4pl
205	float4mi
206	float4um
207	float4abs
209	float4larger
211	float4smaller
212	int4um
213	int2um
216	float8mul
217	float8div
218	float8pl
219	float8mi
220	float8um
221	float8abs
223	float8larger
224	float8smaller
228	dround
229	dtrunc
235	float8
236	float4
237	int2
238	int2
244	timepl
245	timemi
248	intinterval
249	tintervalrel
251	abstimeeq
252	abstimene
253	abstimelt

func_oid_value	func_name
254	abstimegt
255	abstimele
256	abstimege
257	reltimeeq
258	reltimene
259	reltimelt
260	reltimegt
261	reltimele
262	reltimege
263	tintervalsame
264	tintervalct
265	tintervalov
266	tintervalleneq
267	tintervallenne
268	tintervallenlt
269	tintervallengt
270	tintervallenle
271	tintervallenge
273	tintervalend
275	isfinite
279	float48mul
280	float48div
281	float48pl
282	float48mi
283	float84mul
284	float84div
285	float84pl
286	float84mi
287	float4eq
288	float4ne

func_oid_value	func_name
289	float4lt
290	float4le
291	float4gt
292	float4ge
293	float8eq
294	float8ne
295	float8lt
296	float8le
297	float8gt
298	float8ge
299	float48eq
300	float48ne
301	float48lt
302	float48le
303	float48gt
304	float48ge
305	float84eq
306	float84ne
307	float84lt
308	float84le
309	float84gt
310	float84ge
311	float8
312	float4
313	int4
314	int2
316	float8
317	int4
318	float4
319	int4

func_oid_value	func_name
350	btint2cmp
351	btint4cmp
354	btfloat4cmp
355	btfloat8cmp
357	btabstimecmp
358	btcharcmp
360	bttextcmp
377	cash_cmp
380	btreltimecmp
381	bttintervalcmp
385	regexp_count
386	regexp_count
387	regexp_count
400	hashtext
432	hash_numeric
449	hashint2
450	hashint4
451	hashfloat4
452	hashfloat8
454	hashchar
458	text_larger
459	text_smaller
461	int8out
462	int8um
463	int8pl
464	int8mi
465	int8mul
466	int8div
467	int8eq
468	int8ne

func_oid_value	func_name
469	int8lt
470	int8gt
471	int8le
472	int8ge
474	int84eq
475	int84ne
476	int84lt
477	int84gt
478	int84le
479	int84ge
480	int4
481	int8
482	float8
483	int8
630	regexp_instr
631	regexp_instr
632	regexp_instr
633	regexp_instr
634	regexp_instr
652	float4
654	hashint1_numeric
665	hashint2_numeric
667	hashint16
676	mktinterval
682	hashint4_numeric
714	int2
720	octet_length
721	get_byte
722	set_byte
723	get_bit

func_oid_value	func_name
724	set_bit
740	text_lt
741	text_le
742	text_gt
743	text_ge
754	int8
755	hashint8_numeric
766	int4inc
768	int4larger
769	int4smaller
770	int2larger
771	int2smaller
784	tintervaleq
785	tintervalne
786	tintervallt
787	tintervalgt
788	tintervalle
789	tintervalge
792	btint12cmp
793	btint14cmp
794	btint18cmp
795	btint116cmp
796	btint1numericcmp
797	btint21cmp
798	btint216cmp
799	btint2numericcmp
800	btint41cmp
801	btint416cmp
802	btint4numericcmp
803	btint81cmp

func_oid_value	func_name
804	btint816cmp
805	btint8numericcmp
837	int82pl
838	int82mi
839	int82mul
840	int82div
841	int28pl
842	btint8cmp
846	cash_mul_flt4
847	cash_div_flt4
848	flt4_mul_cash
849	position
852	int48eq
853	int48ne
854	int48lt
855	int48gt
856	int48le
857	int48ge
860	bpchar
862	int4_mul_cash
863	int2_mul_cash
864	cash_mul_int4
865	cash_div_int4
866	cash_mul_int2
867	cash_div_int2
868	strpos
870	lower
871	upper
877	substr
883	substr

func_oid_value	func_name
888	cash_eq
889	cash_ne
890	cash_lt
891	cash_le
892	cash_gt
893	cash_ge
894	cash_pl
895	cash_mi
896	cash_mul_flt8
897	cash_div_flt8
898	cashlarger
899	cashsmaller
919	flt8_mul_cash
935	cash_words
936	substring
937	substring
940	mod
941	mod
942	int28mi
943	int28mul
944	char
945	int8mod
947	mod
948	int28div
949	hashint8
1026	timezone
1048	bpchareq
1049	bpcharlt
1050	bpcharle
1051	bpchargt

func_oid_value	func_name
1052	bpcharge
1053	bpcharne
1063	bpchar_larger
1064	bpchar_smaller
1078	bpcharcmp
1080	hashbpchar
1102	time_lt
1103	time_le
1104	time_gt
1105	time_ge
1106	time_ne
1107	time_cmp
1116	regexp_replace
1117	regexp_replace
1118	regexp_replace
1119	regexp_replace
1145	time_eq
1152	timestamptz_eq
1153	timestamptz_ne
1154	timestamptz_lt
1155	timestamptz_le
1156	timestamptz_ge
1157	timestamptz_gt
1158	to_timestamp
1159	timezone
1162	interval_eq
1163	interval_ne
1164	interval_lt
1165	interval_le
1166	interval_ge

func_oid_value	func_name
1167	interval_gt
1168	interval_um
1169	interval_pl
1170	interval_mi
1172	date_part
1173	timestamptz
1177	interval
1180	abstime
1188	timestamptz_mi
1194	reltime
1195	timestamptz_smaller
1196	timestamptz_larger
1197	interval_smaller
1198	interval_larger
1199	age
1200	interval
1218	date_trunc
1219	int8inc
1230	int8abs
1236	int8larger
1237	int8smaller
1238	texticregexeq
1239	texticregexne
1246	charlt
1251	int4abs
1253	int2abs
1254	textregexeq
1256	textregexne
1271	overlaps
1273	date_part

func_oid_value	func_name
1274	int84pl
1275	int84mi
1276	int84mul
1277	int84div
1278	int48pl
1279	int48mi
1280	int48mul
1281	int48div
1282	quote_ident
1283	quote_literal
1289	quote_nullable
1299	now
1304	overlaps
1308	overlaps
1309	overlaps
1310	overlaps
1311	overlaps
1314	timestamptz_cmp
1315	interval_cmp
1316	time
1326	interval_div
1342	round
1343	trunc
1352	timetz_eq
1353	timetz_ne
1354	timetz_lt
1355	timetz_le
1356	timetz_ge
1357	timetz_gt
1358	timetz_cmp

func_oid_value	func_name
1359	timestamptz
1370	interval
1373	isfinite
1374	octet_length
1375	octet_length
1377	time_larger
1378	time_smaller
1379	timetz_larger
1380	timetz_smaller
1384	date_part
1385	date_part
1389	isfinite
1390	isfinite
1394	abs
1395	abs
1396	abs
1397	abs
1398	abs
1419	time
1481	tinterval
1581	biteq
1582	bitne
1592	bitge
1593	bitgt
1594	bitle
1595	bitlt
1596	bitcmp
1608	degrees
1618	interval_mul
1620	ascii

func_oid_value	func_name
1621	chr
1622	repeat
1623	similar_escape
1624	mul_d_interval
1633	texticlike
1634	texticnlike
1637	like_escape
1656	bpcharicregexeq
1657	bpcharicregexne
1658	bpcharregexeq
1659	bpcharregexne
1660	bpchariclike
1661	bpcharicnlike
1666	varbiteq
1667	varbitne
1668	varbitge
1669	varbitgt
1670	varbitle
1671	varbitlt
1672	varbitcmp
1673	bitand
1674	bitor
1675	bitxor
1676	bitnot
1677	bitshiftleft
1678	bitshiftright
1679	bitcat
1680	substring
1682	octet_length
1683	bit

func_oid_value	func_name
1684	int4
1685	bit
1687	varbit
1688	time_hash
1690	time_mi_time
1691	boolle
1692	boolge
1693	btboolcmp
1696	timetz_hash
1697	interval_hash
1698	position
1699	substring
1702	numeric_out
1703	numeric
1704	numeric_abs
1705	abs
1706	sign
1707	round
1709	trunc
1710	trunc
1711	ceil
1712	floor
1718	numeric_eq
1719	numeric_ne
1720	numeric_gt
1721	numeric_ge
1722	numeric_lt
1723	numeric_le
1724	numeric_add
1725	numeric_sub

func_oid_value	func_name
1726	numeric_mul
1727	numeric_div
1728	mod
1729	numeric_mod
1740	numeric
1742	numeric
1743	numeric
1744	int4
1745	float4
1746	float8
1747	time_pl_interval
1748	time_mi_interval
1749	timetz_pl_interval
1750	timetz_mi_interval
1752	trunc
1753	trunc
1764	numeric_inc
1766	numeric_smaller
1767	numeric_larger
1769	numeric_cmp
1771	numeric_uminus
1781	numeric
1782	numeric
1783	int2
1810	bit_length
1811	bit_length
1812	bit_length
1840	int2_sum
1841	int4_sum
1842	int8_sum

func_oid_value	func_name
1845	to_ascii
1846	to_ascii
1848	interval_pl_time
1850	int28eq
1851	int28ne
1852	int28lt
1853	int28gt
1854	int28le
1855	int28ge
1856	int82eq
1857	int82ne
1858	int82lt
1859	int82gt
1860	int82le
1861	int82ge
1874	btint161cmp
1875	btint162cmp
1876	btint164cmp
1877	btint168cmp
1878	btnumericint1cmp
1879	btnumericint2cmp
1880	btnumericint4cmp
1881	btnumericint8cmp
1882	btint16cmp
1892	int2and
1893	int2or
1894	int2xor
1895	int2not
1896	int2shl
1897	int2shr

func_oid_value	func_name
1898	int4and
1899	int4or
1900	int4xor
1901	int4not
1902	int4shl
1903	int4shr
1904	int8and
1905	int8or
1906	int8xor
1907	int8not
1908	int8shl
1909	int8shr
1910	int8up
1911	int2up
1912	int4up
1913	float4up
1914	float8up
1915	numeric_uplus
1946	encode
1961	timestamp
1967	timestamptz
1968	time
1969	timetz
1973	div
1980	numeric_div_trunc
2009	like_escape
2012	substring
2013	substring
2014	position
2020	date_trunc

func_oid_value	func_name
2021	date_part
2024	timestamp
2025	timestamp
2031	timestamp_mi
2032	timestamp_pl_interval
2033	timestamp_mi_interval
2035	timestamp_smaller
2036	timestamp_larger
2038	timezone
2039	timestamp_hash
2041	overlaps
2042	overlaps
2043	overlaps
2044	overlaps
2045	timestamp_cmp
2046	time
2048	isfinite
2052	timestamp_eq
2053	timestamp_ne
2054	timestamp_lt
2055	timestamp_le
2056	timestamp_ge
2057	timestamp_gt
2058	age
2069	timezone
2070	timezone
2073	substring
2074	substring
2075	bit
2076	int8

func_oid_value	func_name
2089	to_hex
2090	to_hex
2160	text_pattern_lt
2161	text_pattern_le
2163	text_pattern_ge
2164	text_pattern_gt
2166	bttext_pattern_cmp
2167	ceiling
2174	bpchar_pattern_lt
2175	bpchar_pattern_le
2177	bpchar_pattern_ge
2178	bpchar_pattern_gt
2180	btbpchar_pattern_cmp
2188	btint48cmp
2189	btint84cmp
2190	btint24cmp
2191	btint42cmp
2192	btint28cmp
2193	btint82cmp
2194	btfloat48cmp
2195	btfloat84cmp
2308	ceil
2309	floor
2310	sign
2320	ceiling
2515	booland_statefunc
2516	boolor_statefunc
2547	interval_pl_timetz
2548	interval_pl_timestamp
2557	bool

func_oid_value	func_name
2558	int4
2765	regexp_split_to_table
2766	regexp_split_to_table
2805	int8inc_float8_float8
2906	timestamptypmodout
2908	timestamptztypmodout
2910	timetypmodout
2912	timetztypmodout
2996	int8_sum_to_int8
3032	get_bit
3033	set_bit
3062	reverse
3167	instr
3168	instr
3169	instr
3170	multiply
3171	multiply
3175	lengthb
3176	lengthb
3177	int8_bool
3178	bool_int8
3180	int2_bool
3181	bool_int2
3182	substring_inner
3183	substring_inner
3226	timestamp_diff
3227	timestamp_diff
3343	int8_mul_cash
3344	cash_mul_int8
3345	cash_div_int8

func_oid_value	func_name
3822	cash_div_cash
3922	int4range_subdiff
3923	int8range_subdiff
3924	numrange_subdiff
3925	daterange_subdiff
3929	tsrange_subdiff
3930	tstzrange_subdiff
4162	varchar_date
4163	bpchar_date
4164	text_date
4166	int2_text
4167	int4_text
4168	int8_text
4169	float4_text
4170	float8_text
4171	numeric_text
5580	smalldatetime_eq
5581	smalldatetime_ne
5582	smalldatetime_lt
5583	smalldatetime_le
5584	smalldatetime_ge
5585	smalldatetime_gt
5586	smalldatetime_cmp
5587	smalldatetime_hash
5809	b_db_last_day
5810	b_db_last_day
5811	b_db_last_day
5816	b_db_last_day
5858	weekofyear
5859	weekofyear

func_oid_value	func_name
5860	weekofyear
5861	weekofyear
6407	int16
6408	int2
6409	int16
6410	int4
6411	int16
6412	int8
6413	int16
6414	float8
6415	int16
6416	float4
6419	int16
6420	int16_bool
6421	int16
6422	numeric
6423	int16eq
6424	int16ne
6425	int16lt
6426	int16le
6427	int16gt
6428	int16ge
6429	int16pl
6430	int16mi
6431	int16mul
6432	int16div
6433	numeric
6434	numeric_bool
6438	int21gt
6439	int21le

func_oid_value	func_name
6440	int21ge
6441	int216eq
6442	int216ne
6443	int216lt
6444	int216gt
6445	int216le
6446	int216ge
6447	int2numericeq
6448	int2numericne
6449	int2numericlt
6450	int2numericgt
6451	int2numericle
6452	int2numericge
6453	int41eq
6454	int41ne
6455	int41lt
6456	int41gt
6457	int41le
6458	int41ge
6459	int416eq
6460	int416ne
6461	int416lt
6462	int416gt
6463	int416le
6464	int416ge
6465	int4numericeq
6466	int4numericne
6467	int4numericlt
6468	int4numericgt
6469	int4numericle

func_oid_value	func_name
6470	int4numericge
6471	int81eq
6472	int81ne
6473	int81lt
6474	int81gt
6475	int81le
6476	int81ge
6477	int816eq
6478	int816ne
6479	int816lt
6480	int816gt
6481	int816le
6482	int816ge
6483	int8numericeq
6484	int8numericne
6485	int8numericlt
6486	int8numericgt
6487	int8numericle
6488	int8numericge
6539	int21eq
6540	int21ne
6578	b_timestampdiff
6579	b_timestampdiff
6582	b_timestampdiff
6583	b_timestampdiff
6584	b_timestampdiff
6585	b_timestampdiff
6586	b_timestampdiff
6587	b_timestampdiff
6588	b_timestampdiff

func_oid_value	func_name
6589	b_timestampdiff
6590	b_timestampdiff
6591	b_timestampdiff
6592	b_timestampdiff
6593	b_timestampdiff
6594	b_timestampdiff
6595	b_timestampdiff
6635	int21lt
6814	int12eq
6815	numericint1eq
6853	int168ge
7747	numericint2le
7748	numericint2ge
7749	numericint4eq
7750	numericint4ne
7751	numericint4lt
7752	numericint4gt
7753	numericint4le
7754	numericint4ge
7755	numericint8eq
7756	numericint8ne
7757	numericint8lt
7758	numericint8gt
7759	numericint8le
7760	numericint8ge
7761	int161eq
7762	int161ne
7763	int161lt
8751	int161gt
8752	int161le

func_oid_value	func_name
8753	int161ge
8754	int162eq
8755	int162ne
8756	int162lt
8757	int162gt
8758	int162le
8759	int162ge
8760	int164eq
8761	int164ne
8762	int164lt
8763	int164gt
8764	int164le
8765	int164ge
8766	int168eq
8767	int168ne
8768	int168lt
8769	int168gt
8770	int168le
9011	smalldatetime_smaller
9012	smalldatetime_larger
9558	int12ne
9559	int12lt
9560	int12gt
9561	int12le
9562	int12ge
9563	int14eq
9564	int14ne
9566	int14lt
9567	int14gt
9568	int14le

func_oid_value	func_name
9569	int14ge
9573	int18eq
9574	int18ne
9575	int18lt
9576	int18gt
9584	int18le
9585	int18ge
9586	int116eq
9587	int116ne
9588	int116lt
9589	int116gt
9590	int116le
9591	int116ge
9592	int1numericeq
9593	int1numericne
9594	int1numericlt
9595	int1numericgt
9596	int1numericle
9597	int1numericge
9624	numericint1ne
9625	numericint1lt
9626	numericint1gt
9627	numericint1le
9628	numericint1ge
9629	numericint2eq
9630	numericint2ne
9631	numericint2lt
9632	numericint2gt
9910	substring_index

8 Best Practices

8.1 Best Practices of Table Design

Using Partitioned Tables

Partitioning refers to splitting what is logically one large table into smaller physical pieces based on specific schemes. The table based on the logic is called a partitioned table, and a physical piece is called a partition. Data is stored on these physical partitions, instead of the logical partitioned table. A partitioned table has the following advantages over an ordinary table:

1. High query performance: You can specify partitions when querying partitioned tables, improving query efficiency.
2. High availability: If a certain partition in a partitioned table is faulty, data in the other partitions is still available.
3. Easy maintenance: To fix a partitioned table having a faulty partition, you only need to fix the partition itself.

   GaussDB supports level-1 and level-2 partitioned tables. There are four types of level-1 partitioned tables: range partitioned tables, interval partitioned tables, list partitioned tables, and hash partitioned tables. Level-2 partitioned tables are composed of nine combinations of any two of the following: range partitioned tables, list partitioned tables, and hash partitioned tables.

   - Range partitioned table: Data within a certain range is mapped to each partition. The range is determined by the partition key specified when the partitioned table is created. This partitioning method is most commonly used. The partition key is usually a date. For example, sales data is partitioned by month.
   - Interval partitioned table: a special type of range partitioned tables. Compared with range partitioned tables, interval value definition is added. When no matching partition can be found for an inserted record, a partition can be automatically created based on the interval value.
   - List partitioned table: Key values contained in the data are stored in different partitions, and the data is mapped to each partition in sequence. The key values contained in the partitions are specified when the partitioned table is created.

- Hash partitioned table: Data is mapped to each partition based on the internal hash algorithm. The number of partitions is specified when the partitioned table is created.
- Level-2 partitioned table: a partitioned table obtained by randomly combining range partitioning, list partitioning, and hash partitioning. Both level-1 and level-2 partitions can be defined in the preceding three ways.

Table Compression Level

When creating a table, you can customize the compression level and compression ratio of fields. Compression affects not only data loading but also data query. The **COMPRESSION** parameter specifies the table compression level.

Parameter description:

COMPRESSION specifies the compression level of table data. It determines the compression ratio and time. Generally, the higher the level of compression, the higher the ratio, the longer the time; and the lower the level of compression, the lower the ratio, the shorter the time. The actual compression ratio depends on the distribution mode of table data loaded.

Value range:

- Valid values for row-store tables are **YES** and **NO**, and the default is **NO**.

You can select different compression levels based on **Table 8-1** in different scenarios.

Table 8-1 Application scenarios of compression levels

Compression Level	Application Scenario	Storage Model
YES	Enabling table compression: You are advised not to enable this function because the compression ratio of row-store tables is low.	Row store
NO	Disabling table compression.	Row store

Selecting a Data Type

Use the following principles to select efficient data types:

1. **Select data types that facilitate data calculation.**

   Generally, the calculation of integers (including common comparison calculations, for example, =, >, <, >=, <=, and !=, as well as GROUP BY) is more efficient than that of strings and floating point numbers.

2. **Select data types with a short length.**

   Data types with short length reduce both the data file size and the memory used for computing, improving the I/O and computing performance. For example, use SMALLINT instead of INT, and INT instead of BIGINT.

3. **Use the same data type for a join.**

   You are advised to use the same data type for a join. To join columns with different data types, the database needs to convert them to the same type, which leads to additional performance overheads.

8.2 Best Practices of SQL Queries

Based on the SQL execution mechanism and a large number of practices, SQL statements can be optimized by following certain rules to enable the database to execute SQL statements more quickly and obtain correct results.

- Replace UNION with UNION ALL.

  UNION eliminates duplicate rows while merging two result sets but **UNION ALL** merges the two result sets without deduplication. Therefore, replace **UNION** with **UNION ALL** if you are sure that the two result sets do not contain duplicate rows based on the service logic.

- Add not null to the join columns.

  If there are many **NULL** values in the **JOIN** columns, you can add the filter criterion **IS NOT NULL** to filter data in advance to improve the **JOIN** efficiency.

- Convert NOT IN to NOT EXISTS.

  Nested Loop Anti Join must be used to implement **NOT IN**, and **Hash Anti Join** is required for **NOT EXISTS**. If no NULL value exists in the **JOIN** columns, NOT IN is equivalent to NOT EXISTS. Therefore, if you are sure that no **NULL** value exists, you can convert NOT IN to NOT EXISTS to generate hash join and to improve the query performance.

  The statements for creating a foreign table are as follows:

  ```
  DROP SCHEMA IF EXISTS no_in_to_no_exists_test CASCADE;
  CREATE SCHEMA no_in_to_no_exists_test;
  SET CURRENT_SCHEMA=no_in_to_no_exists_test;
  CREATE TABLE t1(c1 int, c2 int, c3 int);
  CREATE TABLE t2(d1 int, d2 int NOT NULL, d3 int);
  ```

  The statement for implementing the query using NOT IN is as follows:

  ```
  SELECT * FROM t1 WHERE  c1 NOT IN (SELECT d2 FROM t2);
  ```

  The plan is as follows:

  ```
  gaussdb=# EXPLAIN SELECT * FROM t1 WHERE  c1 NOT IN (SELECT d2 FROM t2);
                            QUERY PLAN
  ------------------------------------------------------------------
   Nested Loop Anti Join  (cost=0.00..29749.02 rows=968 width=12)
     Join Filter: ((t1.c1 = t2.d2) OR (t1.c1 IS NULL))
     ->  Seq Scan on t1  (cost=0.00..29.45 rows=1945 width=12)
     ->  Materialize  (cost=0.00..39.17 rows=1945 width=4)
           ->  Seq Scan on t2  (cost=0.00..29.45 rows=1945 width=4)
  (5 rows)
  ```

  Because there is no null value in the **t2.d2** column (the **t2.d2** column is **NOT NULL** in the table definition), the query can be equivalently modified as follows:

  ```
  SELECT * FROM t1 WHERE  NOT EXISTS (SELECT * FROM t2 WHERE t1.c1=t2.d2);
  ```

  The generated plan is as follows:

  ```
  gaussdb=# EXPLAIN SELECT * FROM t1 WHERE  NOT EXISTS (SELECT * FROM t2 WHERE t1.c1=t2.d2);
                            QUERY PLAN
  ------------------------------------------------------------------
  ```



```
 Hash Anti Join  (cost=53.76..99.14 rows=972 width=12)
   Hash Cond: (t1.c1 = t2.d2)
   ->  Seq Scan on t1  (cost=0.00..29.45 rows=1945 width=12)
   ->  Hash  (cost=29.45..29.45 rows=1945 width=4)
         ->  Seq Scan on t2  (cost=0.00..29.45 rows=1945 width=4)
(5 rows)
```

- Use hashagg.

  If the GROUP BY condition exists in the query statement, the generated plan may contain sorting operations, that is, the plan contains the GroupAgg+Sort operator. As a result, the performance is poor. You can set the GUC parameter **work_mem** to increase the available memory and generate a plan with HashAgg to avoid sorting operations and improve performance. For details about how to set **work_mem**, contact the administrator.

- Replace functions with CASE statements.

  The GaussDB Kernel performance greatly deteriorates if a large number of functions are called. In this case, you can change the pushdown functions to CASE statements.

- Do not use functions or expressions for indexes.

  Using functions or expressions for indexes will stop indexing and enable scanning on the full table.

- Do not use operator (!=, <, or >), NULL, OR, or implicit parameter conversion in WHERE clauses.

- Split complex SQL statements.

  You can split an SQL statement into several ones and save the execution result to a temporary table if the SQL statement is too complex to be tuned using the solutions above, including but not limited to the following scenarios:

  - The same subquery is involved in multiple SQL statements of a job and the subquery contains a large amount of data.
  - Incorrect plan cost causes a small hash bucket of subquery. For example, the actual number of rows is 10 million, but only 1000 rows are in hash bucket.
  - Functions such as substr and to_number cause incorrect measures for subqueries containing a large amount of data.
  - BROADCAST subqueries are performed on large tables in multi-DN environment.

For details about optimization, see **Typical SQL Optimization Methods**.

8.3 Best Practices for Permission Configuration

Context

A database may be used by many users, and users are grouped into a database role for easy management. A database role can be regarded as one or a group of database users.

For databases, users and roles are basically the same. The difference is that when CREATE ROLE is used to create a role, no schema with the same name is created and the user does not have the LOGIN permission by default. When CREATE USER is used to create a user, a schema with the same name is automatically created.

By default, the user has the LOGIN permission. That is, a role with the LOGIN permission can be considered to be a user. In service design, you are advised to use a role to manage permissions rather than accessing databases.

Overview

Improper permission configuration may cause permission exploitation. This section describes the functions of each permission role.

Solution

1. **Database user**

   Database users are used to connect databases, access database objects, and run SQL statements. Only an existing database user can be used to connect databases. Therefore, a database administrator must plan a database user for each user who wants to connect to a database.

   Specify at least the following attributes for a database user:

   By default, database users can be classified into two types, as listed in **Table 8-2**.

   Table 8-2 User types

Type	Description
Initial user	Has the highest-level database rights, that is, has all system and object permissions. Initial users are not affected by the settings of the object permissions. This is comparable to the permissions of **root** in a Unix system. For security purposes, you are advised not to operate as an initial user unless necessary. When installing or initializing a database, you can specify the initial username and password. If you do not specify the username, an initial user with the same name as the OS user who installs the database is automatically generated. If no password is specified, the initial user password is empty after the installation. You need to set the initial user password on the GSQL client before performing other operations. Note: For security purposes, remote login to GaussDB Kernel in trust mode is prohibited for all users, and remote login in any mode is prohibited for the initial user.

Type	Description
Common user	By default, a user can access the default database system catalogs (excluding pg_authid, pg_largeobject, pg_user_status, and pg_auth_history) and views and connect to the default database **postgres**, as well as the objects in the public schema, including tables, views, and functions. • You can run CREATE USER and ALTER USER to specify system permissions, or run GRANT ALL PRIVILEGE to grant the SYSADMIN permission. • You can run the GRANT statement to assign object permissions to a common user. • The user can run the GRANT statement to assign other user permissions to a common user.

2. **Database permission types**

   Permissions and roles work together to specify accessible data and executable SQL statements. For details, see **Table 8-3**.

   System permissions are specified by using the CREATE USER/ALTER USER and CREATE ROLE/ALTER ROLE statements and cannot be inherited from roles. The SYSADMIN permission can be granted or revoked by using the GRANT/REVOKE ALL PRIVILEGES statement.

   Table 8-3 Permission types

Type	Description
System permission	System permissions are also regarded as user attributes, which can be specified when a user is being created or modified. System permissions include SYSADMIN, MONADMIN, OPRADMIN, POLADMIN, CREATEDB, CREATEROLE, AUDITADMIN, and LOGIN. They can be specified only by the CREATE USER or ALTER USER statement. System permissions except SYSADMIN, cannot be granted or revoked by the GRANT or REVOKE statement. In addition, system permissions cannot be inherited from roles.
Object permission	Object permissions are operation permissions for tables, views, indexes, sequences, and functions. These permissions include SELECT, INSERT, UPDATE, and DELETE. Only an object owner or system administrator can use the GRANT/REVOKE statement to grant or revoke object permissions.
Roles	A role is a group of permissions. If a role consists of system permissions, these permissions cannot be granted to other users or roles. If a role consists of object permissions, these permissions can be granted to other users or roles.

3. **Database permission model**

a. **System permission model**

- **Default permission mechanism**

Figure 8-1 Permission architecture

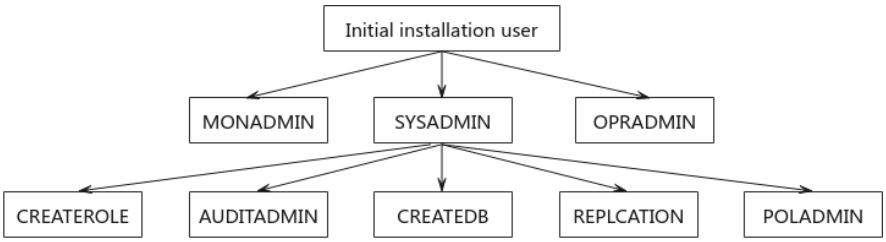
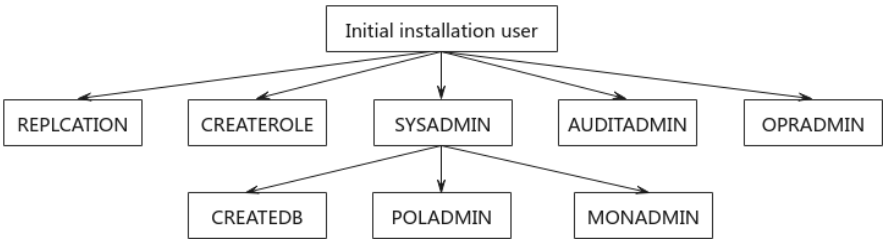


Figure 8-1 shows the permission architecture. In the default permission mechanism, the **sysadmin** user has most permissions.

  - **Initial installation user**: an account automatically generated during cluster installation. This account has the highest permissions in the system and can perform all operations.
  - **SYSADMIN**: system administrator permissions, which are only inferior to those of the initial installation user. By default, the system administrator has the same permissions as the object owner, excluding the permissions of the monitor administrator and O&M administrator.
  - **MONADMIN**: monitoring administrator permissions, including the permissions to access and grant views and functions in the monitoring schema DBE_PERF.
  - **OPRADMIN**: O&M administrator permissions, including the permission to use Roach to perform backup and restoration
  - **CREATEROLE**: security administrator permissions, including the permissions to create, modify, and delete users and roles
  - **AUDITADMIN**: audit administrator permissions, including the permissions to view and maintain database audit logs
  - **CREATEDB**: permission to create databases.
  - **POLADMIN**: security policy administrator permissions, including the permissions to create resource labels, dynamic data masking policies, and unified audit policies.

- **Separation of duties**

Figure 8-2 Separation of duties



- **SYSADMIN**: system administrator permission. The user with this attribute no longer has the permissions to create, modify, delete users or roles, or view or maintain database audit logs.
- **CREATEROLE**: security administrator permissions, including the permissions to create, modify, and delete users and roles
- **AUDITADMIN**: audit administrator permissions, including the permissions to view and maintain database audit logs
- A user or role can have only one of the system permissions SYSADMIN, CREATEROLE, and AUDITADMIN.

b. **Object permission model**

- Object permissions refer to the permissions to perform operations for database objects (such as databases, schemas, and tables), including SELECT, INSERT, UPDATE, DELETE, and CONNECT.
- The permissions vary by object. Object permissions can be granted to users or roles.
- You can use GRANT or REVOKE to grant permissions to a user or revoke them from the user. Object permissions can be inherited by a role.

c. **Role permission model**

GaussDB Kernel provides a group of default roles whose names start with **gs_role_**. These roles are provided to access to specific, typically high-privileged operations. You can grant these roles to other users or roles within the database so that they can use specific functions. These roles should be given with great care to ensure that they are used where they are needed. **Table 8-4** describes the permissions of built-in roles.

Table 8-4 Permissions of built-in roles

Role	Permission
gs_role_copy_files	Permission to run the **copy... to/from filename** command. However, the GUC parameter **enable_copy_server_files** must be set first to enable the function of copying server files.
gs_role_signal_backend	Permission to call the pg_cancel_backend(), pg_terminate_backend(), and pg_terminate_session() functions to cancel or terminate other sessions. However, this role cannot perform operations on sessions of the initial user or users with the **PERSISTENCE** attribute.
gs_role_tablespace	Permission to create a tablespace.

Role	Permission
gs_role_replication	Permission to call logical replication functions, such as kill_snapshot(), pg_create_logical_replication_slot(), pg_create_physical_replication_slot(), pg_drop_replication_slot(), pg_replication_slot_advance(), pg_create_physical_replication_slot_extern(), pg_logical_slot_get_changes(), pg_logical_slot_peek_changes(), pg_logical_slot_get_binary_changes(), and pg_logical_slot_peek_binary_changes().
gs_role_account_lock	Permission to lock and unlock users. However, this role cannot lock or unlock the initial user or users with the **PERSISTENCE** attribute.
gs_role_pldebugger	Permission to debug functions in **dbe_pldebugger**.
gs_role_directory_create	Permission to create directory objects. However, this role needs to enable the GUC parameter **enable_access_server_directory** first.
gs_role_directory_drop	Permission to delete directory objects. However, this role needs to enable the GUC parameter **enable_access_server_directory** first.

4. System permission configuration
   - **Configuring the default permission mechanism**
     - **Initial user**

       The account automatically generated during database installation is called an initial user. The initial user is also the system administrator, monitor administrator, O&M administrator, and security policy administrator. It has the highest permissions in the system and can perform all operations. If the initial username is not specified during installation, the username is the same as the name of the OS user who installs the database. If the password of the initial user is not specified during the installation, the password is empty after the installation. In this case, you need to change the password of the initial user on the gsql client before performing other operations. If the initial user password is empty, you cannot perform other SQL operations, such as upgrade, capacity expansion, and node replacement, except changing the password.

       An initial user bypasses all permission checks. You are advised to use an initial user as a database administrator only for database management other than service running.

     - **System administrator**

       ```
       gaussdb=#CREATE USER u_sysadmin WITH SYSADMIN password '********';
       -- Alternatively, run the following SQL statement when the user already exists:
       gaussdb=#ALTER USER u_sysadmin01 SYSADMIN;
       ```


- **Monitoring administrator**

```
gaussdb=#CREATE USER u_monadmin WITH MONADMIN password '********';
-- Alternatively, run the following SQL statement when the user already exists:
gaussdb=#ALTER USER u_monadmin01 MONADMIN;
```

- **O&M administrator**

```
gaussdb=#CREATE USER u_opradmin WITH OPRADMIN password "xxxxxxxxx";
-- Alternatively, run the following SQL statement when the user already exists:
gaussdb=#ALTER USER u_opradmin01 OPRADMIN;
```

- **Security policy administrator**

```
gaussdb=#CREATE USER u_poladmin WITH POLADMIN password "xxxxxxxxx";
-- Alternatively, run the following SQL statement when the user already exists:
gaussdb=#ALTER USER u_poladmin01 POLADMIN;
```

– **Configuring the separation of duties**

To configure this mode, you need to set the GUC parameter **enableSeparationOfDuty** to **on**. This is a POSTMASTER parameter. After this parameter is modified, you need to restart the database.

```
gs_guc set -Z datanode -N all -I all -c "enableSeparationOfDuty=on"
gs_om -t stop
gs_om -t start
```

The syntax for creating and configuring user permissions is the same as that for default permissions.

5. Role permission configuration

```
-- Create the database test.
gaussdb=#CREATE DATABASE test;
-- Create role1 and user1.
gaussdb=#CREATE ROLE role1 PASSWORD '********';
gaussdb=#CREATE USER user1 PASSWORD '********';
-- Grant the CREATE ANY TABLE permission to role1.
gaussdb=#GRANT CREATE ON DATABASE test TO role1;

-- If role1 is assigned to user1, user1 belongs to group role1 and inherits the permissions of role1 to create schemas in the database test.
gaussdb=#GRANT role1 TO user1;

-- Query user and role information.
gaussdb=#\du role1|user1;
           List of roles
 Role name |  Attributes  | Member of
-----------+--------------+-----------
 role1     | Cannot login | {}
 user1     |              | {role1}
```

Practice Effect

None

9 User-defined Functions

When the database instance is started, the UDF master process is started in addition to the GaussDB main process. To execute a UDF in fenced mode, the UDF master process forks itself to UDF worker processes, and UDF worker processes execute the UDF in fenced mode.

9.1 PL/SQL Functions

PL/SQL is a loadable procedural language.

Functions created using PL/SQL can be used in any place where you can use built-in functions. For example, you can create calculation functions with complex conditions and use them to define operators or use them for index expressions.

SQL is used by most databases as a query language, which is portable and easy to learn. Each SQL statement must be executed independently by a database server.

This means that the client application performs the following processes for each query: send a query to the database server, wait for the query to be received, receive and process the result, perform related calculation, and then send more queries to the server. If the client and the database server are not on the same machine, this process also causes inter-process communication and network load.

PL/SQL enables a whole computing part and a series of queries to be grouped inside a database server. This makes procedural language available and SQL easier to use. In addition, the client/server communication cost is reduced. The advantages of PL/SQL are as follows:

- Extra round-trip communication between clients and servers is eliminated.
- Intermediate results that are not required by clients do not need to be sorted or transmitted between the clients and servers.
- Parsing can be skipped in multiple rounds of queries.

PL/SQL can use all data types, operators, and functions in the SQL statements. The syntax for creating functions using PL/SQL is **CREATE FUNCTION**.

PL/SQL is a loadable procedural language. Its application method is similar to that of **Stored Procedure**. The difference is that **Stored Procedure** has no return value, and PL/SQL functions have return values.

XML data can be used as the input parameter, output parameter, user-defined variable, and return value of a user-defined function.

10 Stored Procedure

10.1 Stored Procedure

In GaussDB, business rules and logics are saved as stored procedures.

A stored procedure is a combination of SQL and PL/SQL. Stored procedures can move the code that executes business rules from applications to databases. Therefore, the code storage can be used by multiple programs at a time.

For details about how to create and call a stored procedure, see **CREATE PROCEDURE**.

The application methods for PL/SQL functions mentioned in **PL/SQL Functions** are similar to those for stored procedures. Unless otherwise specified, the following sections apply to stored procedures and PL/SQL functions.

10.2 Data Types

A data type refers to a value set and an operation set defined on the value set. GaussDB consists of tables, each of which is defined by its own columns. Each column corresponds to a data type. GaussDB uses corresponding functions to perform operations on data based on data types. For example, GaussDB can perform operations such as addition, subtraction, multiplication, and division on numeric data.

XML data can be used as input parameters, output parameters, user-defined variables, and return values of stored procedures, as well as stored procedures that support autonomous transactions.

10.2.1 User-defined Subtypes

In PL/SQL, SUBTYPE can be used to create its own subtypes. The base type can be any basic type or user-defined type. Subtypes can provide data type compatibility, display the intended use of data items of that type, and detect values that are out of range.

The basic definition syntax of SUBTYPE is as follows:


```
SUBTYPE subtype_name IS base_type  [ { ( precision [, scale ] ) | RANGE low_value .. high_value } ] [ NOT NULL ]
```

Parameters:

- **SUBTYPE**: SUBTYPE keyword.
- **subtype_name**: name of the subtype to be defined.
- **base_type**: base type, which can be the base type or user-defined type based on which the subtype is created.
- **precision [, scale]**: typmod constraints that can be added.
- **RANGE low_value .. high_value**: range constraints that can be added.
- **NOT NULL**: NOT NULL constraints that can be added.

NOTE

- **base_type** supports basic types, user-defined data types, and subtypes.
- If the typmod constraint is specified, when a value is assigned to a variable of the SUBTYPE type, the system checks whether the value complies with the typmod constraint and whether the specified typmod constraint can be consistent with the base type.
- Range constraints can be specified only for the int types, including TINYINT, SMALLINT, MEDIUMINT, INTEGER, BINARY_INTEGER, BIGINT, and INT. If the range constraints are specified, the system checks whether the value is within the specified range when assigning a value to a subtype variable.
- The upper and lower limits of RANGE cannot exceed INT64.
- Currently, the RANGE constraints cannot be used when a variable is created.
- The constraint status is updated when the SUBTYPE type is nested or a variable of the SUBTYPE type is created.
- If **NOT NULL** is specified, variables of the SUBTYPE type must be initialized and cannot be assigned **NULL** values.
- The SUBTYPE type can be used as the input and output parameter types of stored procedures.
- Subtype variables can be specified using %type and %rowtype. %type supports all base types, and %rowtype supports tables but does not support sets.
- Subtypes support base type constructors.
- The specified character set is not supported.
- This command can be used only in A-compatible database.
- If a database is upgraded from a version that does not support subtypes to a version that supports subtypes and the upgrade is not committed, subtypes cannot be used.

Example 1: An unconstrained subtype is used.

```
gaussdb=# DECLARE
    SUBTYPE sint IS INT;
    a sint := 2147483647;
 BEGIN
    DBE_OUTPUT.PUT_LINE('a = ' || a);
 END;
 /
a = 2147483647
ANONYMOUS BLOCK EXECUTE
```

Example 2: Subtypes support typmod, range, and NOT NULL constraints.

```
-- typmod constraint
gaussdb=# DECLARE
    SUBTYPE sdec IS DECIMAL(3,2) NOT NULL;
    a sdec := 1.1;
```



```
    b sdec(5,2) := 322.1;
 BEGIN
    DBE_OUTPUT.PUT_LINE('a = ' || a);
    DBE_OUTPUT.PUT_LINE('b = ' || b);
 END;
 /
a = 1.10
b = 322.10
ANONYMOUS BLOCK EXECUTE
-- NOT NULL constraint
gaussdb=# DECLARE
    SUBTYPE sint IS INT NOT NULL;
    a sint;
 BEGIN
    NULL;
 END;
 /
ERROR:  variables declared as NOT NULL must have a default value.
CONTEXT:  compilation of PL/pgSQL function "inline_code_block" near line 2
gaussdb=# DECLARE
    SUBTYPE age IS BINARY_INTEGER RANGE 0..100 NOT NULL;
    a age := 18;
    b age := 20;
 BEGIN
    DBE_OUTPUT.PUT_LINE('Age of a:' || a);
    DBE_OUTPUT.PUT_LINE('Age of b:' || b);
 END;
 /
Age of a: 18
Age of b: 20
ANONYMOUS BLOCK EXECUTE
```

Example 3: Subtypes nest user-defined data types and use the base type constructor.

```
gaussdb=# DECLARE
    TYPE arrint IS VARRAY(10) OF INTEGER;
    SUBTYPE sarrint IS arrint;
    -- a sarrint := sarrint(1,2,3,4): An error is reported. Only the base type constructor is supported.
    a sarrint := arrint(1,2,3,4);
 BEGIN
    FOR i IN 1..4 LOOP
       DBE_OUTPUT.PUT_LINE(a(i));
    END LOOP;
 END;
 /
1
2
3
4
ANONYMOUS BLOCK EXECUTE
```

Example 4: Subtypes support self-nesting, and the constraint condition is updated.

```
-- range constraint
gaussdb=# DECLARE
    SUBTYPE sint IS INTEGER RANGE 10..99;
    SUBTYPE ssint IS sint RANGE 0..9;
    a sint := 50;
    b ssint := 5;
 BEGIN
    DBE_OUTPUT.PUT_LINE('a = ' || a);
    DBE_OUTPUT.PUT_LINE('b = ' || b);
 END;
 /
a = 50
b = 5
ANONYMOUS BLOCK EXECUTE
```



```
-- typmod constraint
gaussdb=# DECLARE
    SUBTYPE word IS VARCHAR2(5);
    SUBTYPE sentence IS word(50);
    a word := 'Tom';
    b sentence := 'Tom and Jerry';
    c sentence(8) := 'Mountain';
 BEGIN
    DBE_OUTPUT.PUT_LINE('a = ' || a);
    DBE_OUTPUT.PUT_LINE('b = ' || b);
    DBE_OUTPUT.PUT_LINE('c = ' || c);
 END;
 /
a = Tom
b = Tom and Jerry
c = Mountain
ANONYMOUS BLOCK EXECUTE
```

Example 5: User-defined data types nest subtypes.

```
gaussdb=# DECLARE
    SUBTYPE sint IS BINARY_INTEGER RANGE 0..99;
    TYPE tabint IS TABLE OF sint;
    a tabint := tabint();
 BEGIN
    a.EXTEND(10);
    a(1) := 50;
 END;
 /
ANONYMOUS BLOCK EXECUTE
```

10.3 Data Type Conversion

Certain data types in the database support implicit data type conversions, such as assignments and parameters called by functions. For other data types (for example, INT and composite types), you can use the type conversion functions provided by GaussDB, such as the **CAST** function, to forcibly convert them.

Table 10-1 lists common implicit data type conversions in GaussDB.

NOTICE

The valid value range of **DATE** supported by GaussDB is from 4713 BC to 294276 AD.

Table 10-1 Implicit data type conversions

Raw Data Type	Target Data Type	Remarks
CHAR	VARCHAR2	-
CHAR	NUMBER	Raw data must consist of digits.
CHAR	DATE	Raw data cannot exceed the valid date range.
CHAR	RAW	-

Raw Data Type	Target Data Type	Remarks
CHAR	CLOB	-
VARCHAR2	CHAR	-
VARCHAR2	NUMBER	Raw data must consist of digits.
VARCHAR2	DATE	Raw data cannot exceed the valid date range.
VARCHAR2	CLOB	-
NUMBER	CHAR	-
NUMBER	VARCHAR2	-
DATE	CHAR	-
DATE	VARCHAR2	-
RAW	CHAR	-
RAW	VARCHAR2	-
CLOB	CHAR	-
CLOB	VARCHAR2	-
CLOB	NUMBER	Raw data must consist of digits.
INT4	CHAR	-
INT4	BOOLEAN	-
BOOLEAN	INT4	-

10.4 Arrays, Collections, and Records

10.4.1 Arrays

Use of Array Types

Before the use of arrays, an array type needs to be defined:

Define an array type immediately after the **AS** keyword in a stored procedure. The definition method is as follows:

```
TYPE array_type IS VARRAY(size) OF data_type;
```

Related parameters are as follows:

- **array_type**: indicates the name of the array type to be defined.
- **VARRAY**: indicates the array type to be defined.

- **size**: indicates the maximum number of members in the array type to be defined. The value is a positive integer.
- **data_type**: indicates the types of members in the array type to be created.

NOTE

- In GaussDB, an array automatically increases. If an access violation occurs, a **NULL** value will be returned, and no error message will be reported.
- The scope of an array type defined in a stored procedure takes effect only in this storage process.
- The **size** supports only the syntax and does not support the function.
- **data_type** can also be the record type defined in a stored procedure (anonymous blocks are not supported) or set type, but cannot be the array or cursor type defined in the stored procedure.
- When **data_type** is collection, multi-dimensional arrays are not supported.
- NOT NULL syntax is not supported.
- The constructors of the array type can be used only in O-compatible mode.
- The constructor of the array type cannot be used as the default value of a function or stored procedure parameter.
- When **data_type** is set to a type that can define the length and precision, such as varchar and numeric, if a varray array is created in a package and invoked outside the package, the restrictions for the length or precision become invalid.

GaussDB supports access to array elements by using parentheses, and it also supports the **extend**, **count**, **first**, **last**, **prior**, **exists**, **trim**, **next**, and **delete** functions.

NOTE

- If a stored procedure contains a DML statement (such as **SELECT**, **UPDATE**, **INSERT**, or **DELETE**), you are advised to use square brackets to access array elements. Using parentheses will access arrays by default. If no array exists, function expressions will be identified.
- Exercise caution when using the **DELETE** statement to delete a single element. Otherwise, the element sequence may be incorrect.
- When the CLOB size is greater than 1 GB, the table of type, record type, and CLOB cannot be used in the input or output parameter, cursor, or raise info in a stored procedure.

Examples

```
-- Perform array operations in the stored procedure.
gaussdb=# CREATE OR REPLACE PROCEDURE array_proc AS
DECLARE
    TYPE ARRAY_INTEGER IS VARRAY(1024) OF INTEGER;--Define the array type.
    ARRINT ARRAY_INTEGER: = ARRAY_INTEGER();  --Declare the variable of the array type.
BEGIN
    ARRINT.extend(10);
    FOR I IN 1..10 LOOP
        ARRINT(I) := I;
    END LOOP;
    DBE_OUTPUT.PRINT_LINE(ARRINT.COUNT);
    DBE_OUTPUT.PRINT_LINE(ARRINT(1));
    DBE_OUTPUT.PRINT_LINE(ARRINT(10));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.FIRST));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.LAST));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.NEXT(ARRINT.FIRST)));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.PRIOR(ARRINT.LAST)));
    ARRINT.TRIM();
```



```
    IF ARRINT.EXISTS(10) THEN
        DBE_OUTPUT.PRINT_LINE('Exist 10th element');
    ELSE
        DBE_OUTPUT.PRINT_LINE('Not exist 10th element');
    END IF;
    DBE_OUTPUT.PRINT_LINE(ARRINT.COUNT);
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.FIRST));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.LAST));
    ARRINT.DELETE();
END;
/

-- Call the stored procedure.
gaussdb=# CALL array_proc();

-- Delete the stored procedure.
gaussdb=# DROP PROCEDURE array_proc;
```

10.4.1.1 Use of Array Types

Before the use of arrays, an array type needs to be defined:

Define an array type immediately after the **AS** keyword in a stored procedure. The definition method is as follows:

```
TYPE array_type IS VARRAY(size) OF data_type;
```

Related parameters are as follows:

- **array_type**: indicates the name of the array type to be defined.
- **VARRAY**: indicates the array type to be defined.
- **size**: indicates the maximum number of members in the array type to be defined. The value is a positive integer.
- **data_type**: indicates the types of members in the array type to be created.

NOTE

- In GaussDB, the array automatically increases. If out-of-bounds access occurs, a **NULL** value is returned and no error is reported. After the **varray_compat** parameter is enabled, the index check is added. If out-of-bounds access occurs, an error is reported.
- The scope of an array type defined in a stored procedure takes effect only in this storage process.
- The size information is recorded in the pg_type system catalog. After the **varray_compat** parameter is enabled, the length and index of array operations are checked. If the parameter is disabled, the size information is not used.
- **data_type** can also be the record type defined in a stored procedure (anonymous blocks are not supported) or set type, but cannot be the array or cursor type defined in the stored procedure.
- When **data_type** is collection, multi-dimensional arrays are not supported.
- NOT NULL syntax is not supported.
- The constructors of the array type can be used only in A-compatible mode.
- The constructor of the array type cannot be used as the default value of a function or stored procedure parameter.
- If an array is an element of the set type and **data_type** of the array is set to **varchar** or **numeric** that can define the length and precision, you need to enable the **tableof_elem_constraints** parameter (**behavior_compat_options** is set to **tableof_elem_constraints**) to verify the element length of the array or convert the element to the corresponding precision.
- After the **varray_compat** parameter is enabled (the **behavior_compat_options** parameter is set to **varray_compat**), the array type defined in the anonymous block cannot be used after ROLLBACK is executed or EXCEPTION occurs in the anonymous block.
- After **enable_recordtype_check_strict** is set to **on**, if the member is of the record type and a column of the record type has the not null or default attribute, an error is reported during stored procedure or package compilation.

GaussDB supports access to array elements by using parentheses, and it also supports the **extend**, **count**, **first**, **last**, **prior**, **exists**, **trim**, **next**, and **delete** functions.

NOTE

- If a stored procedure contains a DML statement (such as **SELECT**, **UPDATE**, **INSERT**, or **DELETE**), you are advised to use square brackets to access array elements. Using parentheses will access arrays by default. If no array exists, function expressions will be identified.
- When the CLOB size is greater than 1 GB, the table of type, record type, and CLOB cannot be used in the input or output parameter, cursor, or raise info in a stored procedure.

Examples

```
-- Perform array operations in the stored procedure.
gaussdb=# CREATE OR REPLACE PROCEDURE array_proc AS
DECLARE
    TYPE ARRAY_INTEGER IS VARRAY(1024) OF INTEGER;--Define the array type.
    ARRINT ARRAY_INTEGER: = ARRAY_INTEGER();  --Declare the variable of the array type.
BEGIN
    ARRINT.extend(10);
    FOR I IN 1..10 LOOP
        ARRINT(I) := I;
    END LOOP;
    DBE_OUTPUT.PRINT_LINE(ARRINT.COUNT);
```



```
    DBE_OUTPUT.PRINT_LINE(ARRINT(1));
    DBE_OUTPUT.PRINT_LINE(ARRINT(10));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.FIRST));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.LAST));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.NEXT(ARRINT.FIRST)));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.PRIOR(ARRINT.LAST)));
    ARRINT.TRIM();

    IF ARRINT.EXISTS(10) THEN
        DBE_OUTPUT.PRINT_LINE('Exist 10th element');
    ELSE
        DBE_OUTPUT.PRINT_LINE('Not exist 10th element');
    END IF;
    DBE_OUTPUT.PRINT_LINE(ARRINT.COUNT);
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.FIRST));
    DBE_OUTPUT.PRINT_LINE(ARRINT(ARRINT.LAST));
    ARRINT.DELETE();
END;
/

-- Call the stored procedure to display the result.
gaussdb=# CALL array_proc();
10
1
10
1
10
2
9
Not exist 10th element
9
1
9
 array_proc
------------

(1 row)

-- Drop the stored procedure.
gaussdb=# DROP PROCEDURE array_proc;
```


10.4.1.2 Functions Supported by Arrays

NOTE

- The following describes the behavior differences of array functions before and after the GUC parameter **varray_compat** is enabled:
  - If the count, extend, trim, delete, first, last, next, or prior function is applied to an uninitialized array, that is, the array is NULL, the "Reference to uninitialized collection" error is reported after the parameter is enabled. If the parameter is disabled, no error is reported.
  - For the extend(count) and extend(count, idx) functions, if the sum of *count* and the number of existing elements in the array exceeds the defined array size, the "Subscript outside of limit" error is reported after the parameter is enabled. If the parameter is disabled, no error is reported.
  - For the extend(count, idx) function, if the index *idx* is invalid, the "Subscript outside of limit" or "Subscript beyond count" error is reported after the parameter is enabled.
  - For the trim(n) function, if *n* is greater than the number of array elements, the "Subscript beyond count" error is reported after the parameter is enabled. If the parameter is disabled, no error is reported.
  - For the delete(idx) function, if the index *idx* is invalid, the "Subscript outside of limit" or "Subscript beyond count" error is reported after the parameter is enabled. If the parameter is disabled, the original array is returned.
- The following is an example in A-compatible mode. In the function definition description, the content in [] is optional. For example, count[()] can be written as count or count().
- If the error "Cannot change the guc status while in the same session" is reported during the execution of the following example, you need to switch the session and execute the example again.
- In an inner expression, functions of the array type cannot be called in nesting mode.

- extend[([count])]

  Parameter: *count* is of the int4 type.

  Return value: No value is returned.

  Description: Extends one or *count* elements at the end of a varray variable. The default value is **NULL**.

  Example 1: The extend is used to extend one element. The default value is **NULL**.

```
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer();
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.extend;
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$# end;
gaussdb$# /
0
1
ANONYMOUS BLOCK EXECUTE
```

  Example 2: The extend(count) is used to extend the *count* elements. The default value is **NULL**.

```
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
```



```
gaussdb-#     arrint array_integer := array_integer();
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.extend(3);
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$# end;
gaussdb$# /
0
3
ANONYMOUS BLOCK EXECUTE
```

Example 3: When the **varray_compat** parameter is enabled, an error is reported if the array size exceeds the defined size after the extend is used.

```
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer();
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.extend(3);
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.extend(8); -- error
gaussdb$# end;
gaussdb$# /
0
3
ERROR:  Subscript outside of limit
CONTEXT:  PL/SQL function inline_code_block line 8 at assignment
```

Example 4: When the **varray_compat** parameter is enabled, an error is reported if the array is not initialized.

```
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer;
gaussdb-# begin
gaussdb$#     arrint.extend(3);
gaussdb$# end;
gaussdb$# /
ERROR:  Reference to uninitialized collection
CONTEXT:  PL/pgSQL function inline_code_block line 4 at assignment
```

- extend(count, idx)

  Parameters: *idx* and *count* are of the int4 type.

  Return value: No value is returned.

  Description: Extends *count* elements at the end of a varray variable, and the value of the extended element is equal to the value of the given *idx* index element. This function can be used only after **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1**.

  Example 1: The extend(count, idx) is used to extend the element at the *idx* position.

```
gaussdb=# set a_format_version='10c';
SET
gaussdb=# set a_format_dev_version='s1';
SET
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer(1,2);
gaussdb-# begin
gaussdb$#     arrint.extend(2, 1);
gaussdb$#     dbe_output.print_line(arrint.count);
```



```
gaussdb$#     for i in 1..arrint.count loop
gaussdb$#         dbe_output.print_line(arrint(i));
gaussdb$#     end loop;
gaussdb$# end;
gaussdb$# /
4
1
2
1
1
ANONYMOUS BLOCK EXECUTE
```

Example 2: extend(count, idx). If the **varray_compat** parameter is enabled and *idx* is out of range, an error is reported.

```
gaussdb=# set a_format_version='10c';
SET
gaussdb=# set a_format_dev_version='s1';
SET
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer();
gaussdb-# begin
gaussdb$#     arrint.extend(10, 1); -- error index.
gaussdb$# end;
gaussdb$# /
ERROR:  Subscript beyond count
CONTEXT:  PL/SQL function inline_code_block line 5 at assignment
```

Example 3: extend(count, idx). If the **varray_compat** parameter is enabled and the array is not initialized, an error is reported.

```
gaussdb=# set a_format_version='10c';
SET
gaussdb=# set a_format_dev_version='s1';
SET
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer;
gaussdb-# begin
gaussdb$#     arrint.extend(10, 1);
gaussdb$# end;
gaussdb$# /
ERROR:  Reference to uninitialized collection
CONTEXT:  PL/pgSQL function inline_code_block line 4 at assignment
```

- count[()]

  Parameter: none.

  Return value: int4 type.

  Description: Returns the number of elements in an array.

  Example 1: View the number of initialized array elements.

```
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer;
gaussdb-#     len int;
gaussdb-# begin
gaussdb$#     arrint := array_integer(1, 2, 3);
gaussdb$#     len := arrint.count();
gaussdb$#     dbe_output.print_line(len);
gaussdb$# end;
gaussdb$# /
3
ANONYMOUS BLOCK EXECUTE
```


Example 2: When the **varray_compat** parameter is enabled, an error is reported if the array is not initialized.

```
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer;
gaussdb-#     len int;
gaussdb-# begin
gaussdb$#     len := arrint.count();
gaussdb$#     dbe_output.print_line(len);
gaussdb$# end;
gaussdb$# /
ERROR:  Reference to uninitialized collection
CONTEXT:  PL/SQL function inline_code_block line 6 at assignment
```

- trim[(n)]

  Parameter: *n* is of the int4 type.

  Return value: No value is returned.

  Description: Deletes an element space at the end of the array if there is no parameter. Alternatively, deletes a specified number of element spaces at the end of the array if the parameter *n* is specified.

  Example 1: Delete *n* elements from an array without enabling the **varray_compat** parameter.

```
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer(1,2,3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.trim(1);
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$# end;
gaussdb$# /
3
2
ANONYMOUS BLOCK EXECUTE
-- If n is greater than the number of array elements, clear all array elements.
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer(1,2,3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.trim(4);
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$# end;
gaussdb$# /
3
0
ANONYMOUS BLOCK EXECUTE
```

  Example 2: Delete *n* elements from an array (*n* > Number of array elements) without enabling the **varray_compat** parameter.

```
-- If n is greater than the number of array elements, clear all array elements.
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer(1,2,3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.trim(4);
gaussdb$#     dbe_output.print_line(arrint.count);
```



```
gaussdb$# end;
gaussdb$# /
3
0
ANONYMOUS BLOCK EXECUTE
```

Example 3: Delete *n* elements from an array (*n* > Number of array elements) with the **varray_compat** parameter enabled.

```
-- If n is greater than the number of array elements, an error is reported.
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer(1,2,3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count); -- count > 0.
gaussdb$#     arrint.trim(4);
gaussdb$# end;
gaussdb$# /
3
ERROR:  Subscript beyond count
CONTEXT:  PL/SQL function inline_code_block line 6 at assignment
-- An error is reported when the array is not initialized.
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer;
gaussdb-# begin
gaussdb$#     arrint.trim(1);
gaussdb$# end;
gaussdb$# /
ERROR:  Reference to uninitialized collection
CONTEXT:  PL/SQL function inline_code_block line 5 at assignment
```

Example 4: The array is not initialized, and the **varray_compat** parameter is enabled.

```
-- An error is reported when the array is not initialized.
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer;
gaussdb-# begin
gaussdb$#     arrint.trim(1);
gaussdb$# end;
gaussdb$# /
ERROR:  Reference to uninitialized collection
CONTEXT:  PL/SQL function inline_code_block line 5 at assignment
```

- delete[()]

  Parameter: none

  Return value: No value is returned.

  Description: Clears elements in an array.

  Example 1: Clear the elements in the array without enabling the **varray_compat** parameter.

```
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer(1,2,3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.delete();
```



```
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$# end;
gaussdb$# /
3
0
ANONYMOUS BLOCK EXECUTE
-- The array is not initialized.
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer;
gaussdb-# begin
gaussdb$#     arrint.delete();
gaussdb$# end;
gaussdb$# /
ANONYMOUS BLOCK EXECUTE
```

Example 2: Clear the elements in the array with the **varray_compat** parameter enabled.

```
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer(1,2,3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.delete();
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$# end;
gaussdb$# /
3
0
ANONYMOUS BLOCK EXECUTE
-- An error is reported when the array is not initialized.
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer;
gaussdb-# begin
gaussdb$#     arrint.delete();
gaussdb$# end;
gaussdb$# /
ERROR:  Reference to uninitialized collection
CONTEXT:  PL/SQL function inline_code_block line 5 at assignment
```

- delete(idx)

  Parameter: int4 type.

  Return value: No value is returned.

  Description: If the specified index *idx* is within the array range, the element with the specified index *idx* is deleted from the array.

  Example 1: The given index *idx* is within the array range.

```
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer(1, 2, 3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.delete(2);
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     for i in 1..arrint.count loop
gaussdb$#         dbe_output.print_line(arrint(i));
gaussdb$#     end loop;
gaussdb$# end;
gaussdb$# /
3
```



```
2
1
3
ANONYMOUS BLOCK EXECUTE
```

Example 2: The specified *idx* index is out of the array range, and the **varray_compat** parameter is not enabled.

```
-- The varray_compat parameter is not enabled.
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer(1, 2, 3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.delete(4);
gaussdb$#     raise info '%', arrint;
gaussdb$# end;
gaussdb$# /
3
INFO:  {1,2,3}
ANONYMOUS BLOCK EXECUTE
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer:= array_integer(1, 2, 3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.delete(11);
gaussdb$#     raise info '%', arrint;
gaussdb$# end;
gaussdb$# /
3
INFO:  {1,2,3}
ANONYMOUS BLOCK EXECUTE

-- An error is reported after the varray_compat parameter is enabled.
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer(1, 2, 3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.delete(4);
gaussdb$#     raise info '%', arrint;
gaussdb$# end;
gaussdb$# /
3
ERROR:  Subscript beyond count
CONTEXT:  PL/SQL function inline_code_block line 6 at assignment

gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer:= array_integer(1, 2, 3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.delete(11);
gaussdb$#     raise info '%', arrint;
gaussdb$# end;
gaussdb$# /
3
ERROR:  Subscript outside of limit
CONTEXT:  PL/SQL function inline_code_block line 6 at assignment
```

Example 3: The specified *idx* index is out of the array range, and the **varray_compat** parameter is enabled.


```
-- An error is reported after the varray_compat parameter is enabled.
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer := array_integer(1, 2, 3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.delete(4);
gaussdb$#     raise info '%', arrint;
gaussdb$# end;
gaussdb$# /
3
ERROR:  Subscript beyond count
CONTEXT:  PL/SQL function inline_code_block line 6 at assignment

gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer:= array_integer(1, 2, 3);
gaussdb-# begin
gaussdb$#     dbe_output.print_line(arrint.count);
gaussdb$#     arrint.delete(11);
gaussdb$#     raise info '%', arrint;
gaussdb$# end;
gaussdb$# /
3
ERROR:  Subscript outside of limit
CONTEXT:  PL/SQL function inline_code_block line 6 at assignment
```

Example 4: When the **varray_compat** parameter is enabled, an error is reported if the array is not initialized.

```
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type array_integer is varray(10) of integer;
gaussdb-#     arrint array_integer;
gaussdb-# begin
gaussdb$#     arrint.delete(2);
gaussdb$# end;
gaussdb$# /
ERROR:  Reference to uninitialized collection
CONTEXT:  PL/pgSQL function inline_code_block line 4 at assignment
```

- first[()]

  Parameter: none.

  Return value: int4 type.

  Description: Returns the index of the first element in an array. If there is no first element, NULL is returned.

  Example:

```
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr := varr('asd','zxc');
gaussdb-# begin
gaussdb$#     raise info 'first is %',v.first();
gaussdb$# end;
gaussdb$# /
INFO:  first is 1
ANONYMOUS BLOCK EXECUTE
-- If the array is not initialized, NULL is returned.
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr;
```



```
gaussdb-# begin
gaussdb$#     raise info 'first is %',v.first;
gaussdb$# end;
gaussdb$# /
INFO:  first is <NULL>
ANONYMOUS BLOCK EXECUTE
-- After the varray_copmat parameter is enabled, an error is reported when the array is not initialized.
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr;
gaussdb-# begin
gaussdb$#     raise info 'first is %',v.first;
gaussdb$# end;
gaussdb$# /
ERROR:  Reference to uninitialized collection
CONTEXT:  PL/pgSQL function inline_code_block line 4 at RAISE
```

- last[()]

  Parameter: none.

  Return value: int4 type.

  Description: Returns the index of the last element in an array. If there is no last element, NULL is returned.

  Example 1:

```
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr := varr('asd','zxc');
gaussdb-# begin
gaussdb$#     raise info 'last is %',v.last();
gaussdb$# end;
gaussdb$# /
INFO:  last is 2
ANONYMOUS BLOCK EXECUTE
-- If the array is not initialized, NULL is returned.
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr;
gaussdb-# begin
gaussdb$#     raise info 'last is %',v.last;
gaussdb$# end;
gaussdb$# /
INFO:  last is <NULL>
ANONYMOUS BLOCK EXECUTE
-- After the varray_copmat parameter is enabled, an error is reported when the array is not initialized.
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr;
gaussdb-# begin
gaussdb$#     raise info 'first is %',v.last;
gaussdb$# end;
gaussdb$# /
ERROR:  Reference to uninitialized collection
CONTEXT:  PL/pgSQL function inline_code_block line 4 at RAISE
```

- prior(idx)

  Parameter: int4 type.

  Return value: int4 type.

  Description: Returns the index of the element before the specified index in an array. If the index of the element cannot be found, NULL is returned.

Example:

```
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr := varr('asd','zxc');
gaussdb-# begin
gaussdb$#     raise info 'prior is %',v.prior(2);
gaussdb$# end;
gaussdb$# /
INFO:  prior is 1
ANONYMOUS BLOCK EXECUTE

-- The index is out of range and is greater than the array range.
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr := varr('asd','zxc','qwe');
gaussdb-# begin
gaussdb$#     raise info 'prior is %',v.prior(10);
gaussdb$# end;
gaussdb$# /
INFO:  prior is 3
ANONYMOUS BLOCK EXECUTE

-- After the varray_copmat parameter is enabled, an error is reported when the array is not initialized.
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr;
gaussdb-# begin
gaussdb$#     raise info 'prior is %',v.prior(2);
gaussdb$# end;
gaussdb$# /
ERROR:  Reference to uninitialized collection
CONTEXT:  PL/pgSQL function inline_code_block line 4 at RAISE
```

- next(idx)

  Parameter: int4 type.

  Return value: int4 type.

  Description: Returns the index of the element following the specified index in an array. If the index of an element cannot be found, NULL is returned.

  Example:

```
gaussdb=# set behavior_compat_options = '';
SET
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr := varr('asd','zxc');
gaussdb-# begin
gaussdb$#     raise info 'next is %',v.next(1);
gaussdb$# end;
gaussdb$# /
INFO:  next is 2
ANONYMOUS BLOCK EXECUTE

-- The index is out of range and is greater than the array range.
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr := varr('asd','zxc','qwe');
gaussdb-# begin
gaussdb$#     raise info 'next  is %',v.next(10);
gaussdb$# end;
gaussdb$# /
INFO:  next  is <NULL>
ANONYMOUS BLOCK EXECUTE
```



```
-- After the varray_copmat parameter is enabled, an error is reported when the array is not initialized.
gaussdb=# set behavior_compat_options = 'varray_compat';
SET
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr;
gaussdb-# begin
gaussdb$#     raise info 'next  is %',v.next(1);
gaussdb$# end;
gaussdb$# /
ERROR:  Reference to uninitialized collection
CONTEXT:  PL/pgSQL function inline_code_block line 4 at RAISE
```

- exists(idx)

  Parameter: int4 type.

  Return value: **true** or **false**, of the Boolean type.

  Description: Checks whether a valid element exists in a specified position.

  Example:

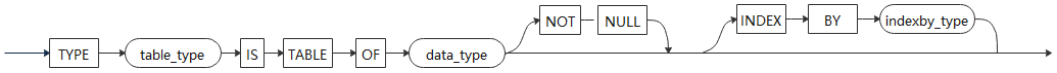
```
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr := varr('asd','zxc');
gaussdb-#     flag bool;
gaussdb-# begin
gaussdb$#     flag := v.exists(1);
gaussdb$#     raise info '%',flag;
gaussdb$#     flag := v.exists(3);
gaussdb$#     raise info '%',flag;
gaussdb$#     flag := v.exists(7);
gaussdb$#     raise info '%',flag;
gaussdb$# end;
gaussdb$# /
INFO:  t
INFO:  f
INFO:  f
ANONYMOUS BLOCK EXECUTE
-- If the array is not initialized, false is returned.
gaussdb=# declare
gaussdb-#     type varr is varray(10) of varchar(3);
gaussdb-#     v varr;
gaussdb-#     flag bool;
gaussdb-# begin
gaussdb$#     flag := v.exists(1);
gaussdb$#     raise info '%',flag;
gaussdb$#     flag := v.exists(3);
gaussdb$#     raise info '%',flag;
gaussdb$#     flag := v.exists(7);
gaussdb$#     raise info '%',flag;
gaussdb$# end;
gaussdb$# /
INFO:  f
INFO:  f
INFO:  f
ANONYMOUS BLOCK EXECUTE
```

10.4.2 Collections

10.4.2.1 Use of Collection Types

Before the use of collections, a collection type must be defined.

Define a collection type immediately after the AS keyword in a stored procedure as follows.

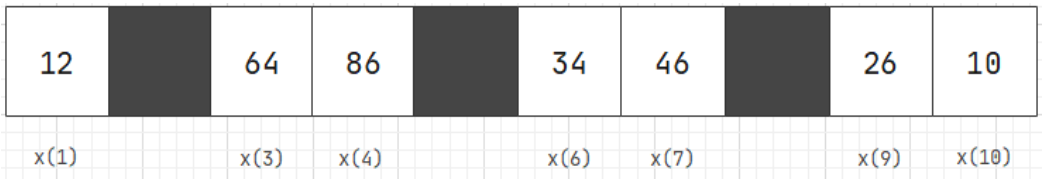


In the preceding information:

- **table_type**: indicates the name of the collection type to be defined.
- **TABLE**: indicates the collection type to be defined.
- **data_type**: indicates the type of members in the collection to be created.
- **indexby_type**: indicates the type of the index set to be created.

Nest-Table Collection Type

Members of a specified data type are stored in a variable-length array. You can use the extend function to expand the storage space and use the trim function to release the storage space. For a collection variable *x* with 10 storage units whose member type is int, members are stored as shown in the following figure.



Members x(2), x(5), and x(8) are invalid, but their storage space is still reserved. You can assign values to them later without re-allocating space.

After a collection type is defined, use table_type as the type name to declare the variable.

```
var_name table_type [:= table_type([v1[,...]])];
```

You can use a type constructor to initialize the variable during or after declaration. If the variable is not initialized, the value of *var_name* is **NULL**.

After the variable is declared and initialized, you can access collection members through an index set or assign values to members. The index set range is [1,*upper*]. The value of *upper* is the size of the current space. If you attempt to access a deleted member, the error message "no data found" will be returned.

NOTE

- In a non-A compatible mode (the value of **sql_compatibility** is not **A**), the collection type cannot be created.
- In GaussDB, a nest-table collection does not automatically increase, and an error is reported when you attempt to access the index set out of the specified range.
- Collections can be defined in schemas, anonymous blocks, stored procedures, user-defined functions, and packages, among which the scopes of the collection types vary.
- **NOT NULL** has no function but only takes effect in the syntax.
- When **data_type** is set to a type that can define the length and precision, such as varchar and numeric, you need to enable the **tableof_elem_constraints** parameter (**behavior_compat_options** must be set to **tableof_elem_constraints**) to verify the length of elements in the collection or convert elements to the corresponding precision.
- If **data_type** is of the array type, the element length verification or precision conversion of the array type is also affected by whether the **tableof_elem_constraints** parameter is enabled.
- The value of the collection type converted from the array type does not support element length verification or precision conversion.
- The value of **data_type** can be a base data type or a record type, collection type, or array type defined in a stored procedure. The ref cursor type is not supported.
- Variables of a collection type cannot be assigned a value of another collection type, even if their member types are the same. For example, *t1* and *t2* are of different collection types defined by **TYPE** *t1* **IS TABLE OF** *int* and **TYPE** *t2* **IS TABLE OF** *int*, respectively. A value of the collection type defined by **TYPE** *t1* **IS TABLE OF** *int* cannot be assigned to *t2*, and vice versa. (This restriction may not take effect when the variables are of the member types because the value assignment logic of variables is affected by the parent type.)
- Only the equal (=) and non-equal operations (<> or !=) between collections are supported. Other relational and arithmetic operations are not supported.
- Use IS [NOT] NULL to compare a collection type with **NULL**. The result of comparison with **NULL** using the equal operator (=) is inaccurate.
- Variables of the collection type can be used as parameters and return values of functions. In this case, the type of the parameters or return values must have been defined in a schema or package.
- When a nest-table collection is used as the input parameter of a function, an array with elements of the same type can be transferred as the input parameter. Multi-dimensional arrays are not supported, and the array index set must start from 1. (This function is outdated and not recommended.) You can run the **set behavior_compat_options = 'disable_rewrite_nesttable'** command to disable the function.
- Operations on XML data are not supported.
- When creating a table, do not use the collection type or any type containing a collection as a column in the table.
- Constructors of the collection type do not support floating point numbers and expressions as indexes.
- For a collection type defined in an anonymous block, after ROLLBACK is executed or EXCEPTION occurs in an anonymous block, the collection type cannot be used.
- If **set enable_recordtype_check_strict** is set to **on;** and the member is of the record type and a column of the record type has the not null or default attribute, an error is reported during stored procedure or package compilation.

GaussDB supports access to collection elements by using parentheses, and it also supports the extend, count, first, last, prior, next, and delete functions.

The collection functions support multiset union, intersect, except all, and distinct.

Index-By Table Collection Type

This collection type stores the index set and corresponding member values in a hash table as key-value pairs. All operations on variables of this type are actually operations on the hash table. Users do not need to expand or release storage space, but only need to assign values or delete members. The operations related to the collection type are described as follows:

1. Type definition

   When defining the index-by table collection type, specify both the member type **data_type** and index set type **indexby_type**. The index set type can only be integer or varchar.

2. Variable declaration and initialization

   After the index-by table collection type is declared, it can be initialized. There are three initialization scenarios: uninitialized, initialized to null, and initialized to specified index set and member values. The effect of uninitialized variables is the same as that of initializing variables to null. If a variable is not initialized or is initialized to null, the variable is not **NULL**. You can assign a value to the variable later. Initializing a variable to specified index set and member values will save the specified index set and member values to the variable as key-value pairs.

3. Variable assignment

   Assignment to variables of the index-by table collection type is classified into member assignment and group assignment. Member assignment allows to assign a value to a member by specifying the index set. If the member does not have a value, the assigned value is used directly. If the member has a value, the member value is updated. Group assignment will clear the original members in the variable and save the new member values. In the group assignment scenario, **NULL** cannot be assigned to variables. Otherwise, an error is reported.

4. Variable value

   You can specify an index set to obtain the member value of the corresponding index in the variable. If the member cannot be found based on the index set, the error message "no data found" is returned.

NOTE

- In a non-A compatible mode (the value of **sql_compatibility** is not **A**), no index-by table collection type can be created.
- Index-by table collection types can be defined in anonymous blocks, stored procedures, user-defined functions, and packages, among which the scopes of the collection types vary. Index-by table collection types cannot be defined in schemas.
- **NOT NULL** has no function but only takes effect in the syntax.
- When **data_type** is set to a type that can define the length and precision, such as varchar and numeric, you need to enable the **tableof_elem_constraints** parameter (**behavior_compat_options** must be set to **tableof_elem_constraints**) to verify the length of elements in the collection or convert elements to the corresponding precision. In the syntax of creating a collection containing elements of the CHAR, VACHAR, NUMERIC, FLOAT, or NUMBER type, the syntax cannot restrict the element range. For example, **create type t1 is table of numeric(5) index by int;** is the same as **type t1 is table of numeric index by int;** and the range restriction does not take effect.
- If **data_type** is of the array type, the element length verification or precision conversion of the array type is also affected by whether the **tableof_elem_constraints** parameter is enabled.
- The value of the collection type converted from the array type does not support element length verification or precision conversion.
- The **data_type** can be a base data type or a record type, collection type, or array type defined in a stored procedure. The ref cursor type is not supported.
- The value of **indexby_type** can only be INTEGER or VARCHAR.
- When **indexby_type** is set to **varchar** and **tableof_elem_constraints** is enabled, the length of the index value is verified when a value is assigned to an index-by table collection type. The verification behavior is not affected by whether **char_coerce_compat** is enabled. If the index length is greater than the defined length, an error is reported. If the **tableof_elem_constraints** parameter is disabled, the index value length is not verified.
- An uninitialized variable of the index-by table collection type is not **NULL**.
- **NULL** cannot be assigned to a variable of the index-by table collection type. Otherwise, an error is reported.
- **NULL** and '' (single quotation marks) cannot be assigned to a variable of the index-by table collection type.
- Variables of the index-by table collection type cannot be assigned values of another index-by table collection type, even if their member type and index set type are the same. For example, *t1* and *t2* are of different collection types defined by **TYPE** *t1* **IS TABLE OF** *int* **index by** *int* and **TYPE** *t2* **IS TABLE OF** *int* **index by** *int*, respectively. A value of the collection type defined by **TYPE** *t1* **IS TABLE OF** *int* **index by** *int* cannot be assigned to *t2*, and vice versa. (This restriction may not take effect when the variables are of the member types because the value assignment logic of variables is affected by the parent type.)
- An index-by table collection type does not support relational and arithmetic operations.
- When a variable of the index-by table collection type is assigned by executing **select... bulk collect into**, the index set must be of the integer type. The index set type cannot be varchar.
- Variables of the index-by table collection type can be used as the parameters and return values of functions. In this case, the type of the parameters or return values must be a collection type defined in the package.
- When an index-by table collection is used as the input parameter of a function, an array with elements of the same type can be transferred as the input parameter. Multi-dimensional arrays are not supported, and the index type cannot be VARCHAR. (This function is outdated and not recommended. You can run the **set behavior_compat_options = 'disable_rewrite_nesttable';** command to disable the function.)

- Currently, the type constructor supports only the collection type and the maximum number of parameters is the same as that of user-defined function parameters. For the index-by table collection type, the index value can only be a constant when the constructor is used.
- Operations on XML data are not supported.
- When creating a table, do not use the collection type or any type containing a collection as a column in the table.
- Constructors of the collection type do not support floating point numbers and expressions as indexes.
- For a collection type defined in an anonymous block, after ROLLBACK is executed or EXCEPTION occurs in an anonymous block, the collection type cannot be used.
- If **set enable_recordtype_check_strict** is set to **on;** and the member is of the record type and a column of the record type has the not null or default attribute, an error is reported during stored procedure or package compilation.

Examples

Example 1: nest-table collection type

```
-- Perform operations on a collection in the stored procedure.
gaussdb=# CREATE OR REPLACE PROCEDURE table_proc AS
DECLARE
    TYPE TABLE_INTEGER IS TABLE OF INTEGER;-- Define a collection type.
    TABLEINT TABLE_INTEGER := TABLE_INTEGER();  -- Declare the variable of the collection type.
BEGIN
    TABLEINT.extend(10);
    FOR I IN 1..10 LOOP
        TABLEINT(I) := I;
    END LOOP;
    DBE_OUTPUT.PRINT_LINE(TABLEINT.COUNT);
    DBE_OUTPUT.PRINT_LINE(TABLEINT(1));
    DBE_OUTPUT.PRINT_LINE(TABLEINT(10));
END;
/
CREATE PROCEDURE
-- Call the stored procedure.
gaussdb=# CALL table_proc();
CALL table_proc();
10
1
10
 table_proc
------------


(1 row)
-- Drop the stored procedure.
gaussdb=# DROP PROCEDURE table_proc;
DROP PROCEDURE
-- Perform operations on a nest-table collection in the stored procedure.
gaussdb=# CREATE OR REPLACE PROCEDURE nest_table_proc AS
DECLARE
    TYPE TABLE_INTEGER IS TABLE OF INTEGER;-- Define a collection type.
    TYPE NEST_TABLE_INTEGER IS TABLE OF TABLE_INTEGER;-- Define a collection type.
    NEST_TABLE_VAR NEST_TABLE_INTEGER := NEST_TABLE_INTEGER(); --Declare a variable of the nest-table collection type.
BEGIN
    NEST_TABLE_VAR.extend(10);
    FOR I IN 1..10 LOOP
        NEST_TABLE_VAR(I) := TABLE_INTEGER();
        NEST_TABLE_VAR(I).extend(10);
        NEST_TABLE_VAR(I)(I) := I;
    END LOOP;
    DBE_OUTPUT.PRINT_LINE(NEST_TABLE_VAR.COUNT);
    DBE_OUTPUT.PRINT_LINE(NEST_TABLE_VAR(1)(1));
    DBE_OUTPUT.PRINT_LINE(NEST_TABLE_VAR(10)(10));
```



```
END;
/
CREATE PROCEDURE
-- Call the stored procedure.
gaussdb=# CALL nest_table_proc();
10
1
10
 nest_table_proc
-----------------

(1 row)
-- Drop the stored procedure.
gaussdb=# DROP PROCEDURE nest_table_proc;
DROP PROCEDURE
```

Example 2: index-by table collection type

```
-- Perform operations on an index-by table collection in the stored procedure.
gaussdb=# CREATE OR REPLACE PROCEDURE index_table_proc AS
DECLARE
    TYPE TABLE_INTEGER IS TABLE OF INTEGER INDEX BY INTEGER; -- Define a collection type.
    TYPE TABLE_VARCHAR IS TABLE OF INTEGER INDEX BY VARCHAR; -- Define a collection type.
    TABLEINT_01 TABLE_INTEGER;                          -- Declare a variable of the collection type, which is
not initialized.
    TABLEINT_02 TABLE_INTEGER := TABLE_INTEGER();          -- Declare a variable of the collection type.
The initial value is null.
    TABLEINT_03 TABLE_INTEGER := TABLE_INTEGER(2=>3,3=>4);   -- Declare a variable of the collection
type and initialize it to the specified value.
    RES INTEGER;
BEGIN
    FOR I IN 1..10 LOOP
        TABLEINT_01(I) := I;     -- Assign values to members.
        TABLEINT_02(I) := I + 1; -- Assign values to members.
    END LOOP;
    TABLEINT_01 := TABLEINT_02;  -- Group assignment
    RES := TABLEINT_03(2);       -- Return the collection values.
    DBE_OUTPUT.PRINT_LINE(RES);
    DBE_OUTPUT.PRINT_LINE(TABLEINT_01(1));
    DBE_OUTPUT.PRINT_LINE(TABLEINT_01(10));
END;
/
CREATE PROCEDURE
-- Call the stored procedure.
gaussdb=# CALL index_table_proc();
3
2
11
 index_table_proc
------------------

(1 row)
-- Drop the stored procedure.
gaussdb=# DROP PROCEDURE index_table_proc;
DROP PROCEDURE
-- Perform operations on a nest-table collection in the stored procedure.
gaussdb=# CREATE OR REPLACE PROCEDURE nest_table_proc AS
DECLARE
    TYPE TABLE_INTEGER IS TABLE OF INTEGER INDEX BY INTEGER; -- Define a collection type.
    TYPE NEST_TABLE_INTEGER IS TABLE OF TABLE_INTEGER INDEX BY INTEGER;-- Define a collection
type.
    NEST_TABLE_VAR NEST_TABLE_INTEGER;                          -- Declare variables of the nest-table
collection type.
BEGIN
    FOR I IN 1..10 LOOP
        NEST_TABLE_VAR(I)(I) := I;
    END LOOP;
    DBE_OUTPUT.PRINT_LINE(NEST_TABLE_VAR.COUNT);
    DBE_OUTPUT.PRINT_LINE(NEST_TABLE_VAR(1)(1));
    DBE_OUTPUT.PRINT_LINE(NEST_TABLE_VAR(10)(10));
```



```
END;
/
CREATE PROCEDURE
-- Call the stored procedure.
gaussdb=# CALL nest_table_proc();
CALL nest_table_proc();
10
1
10
 nest_table_proc
-----------------

(1 row)
-- Drop the stored procedure.
gaussdb=# DROP PROCEDURE nest_table_proc;
DROP PROCEDURE
```

10.4.2.2 Collection Functions

Collection Operators

- =

  Parameter: nesttable

  Return value: **TRUE** or **FALSE**, of the Boolean type.

  Description: Checks whether two collections are equal.

  Example:

  ```
  gaussdb=# declare
  gaussdb-#     type nest is table of int;
  gaussdb-#     a nest := nest(1,2);
  gaussdb-#     b nest := nest(1,2);
  gaussdb-#     flag bool;
  gaussdb-# begin
  gaussdb$#     flag := a = b;
  gaussdb$#     raise info '%', flag;
  gaussdb$# end;
  gaussdb$# /
  INFO:  t
  ANONYMOUS BLOCK EXECUTE
  ```

- <>

  Parameter: nesttable

  Return value: **TRUE** or **FALSE**, of the Boolean type.

  Description: Checks whether two collections are not equal.

  Example:

  ```
  gaussdb=# declare
  gaussdb-#     type nest is table of int;
  gaussdb-#     a nest := nest(1,2);
  gaussdb-#     b nest := nest(1,2);
  gaussdb-#     flag bool;
  gaussdb-# begin
  gaussdb$#     flag := a <> b;
  gaussdb$#     raise info '%', flag;
  gaussdb$# end;
  gaussdb$# /
  INFO:  f
  ANONYMOUS BLOCK EXECUTE
  ```

- IS [NOT] NULL

  Parameter: nesttable

  Return value: **TRUE** or **FALSE**, of the Boolean type.

Description: Determines whether a collection is **NULL**.

Example:

```
gaussdb=# declare
gaussdb-#     type nest is table of int;
gaussdb-#     a nest;
gaussdb-#     b nest := NULL;
gaussdb-#     c nest := nest(1,2);
gaussdb-#     flag bool;
gaussdb-# begin
gaussdb$#     flag := a IS NULL;
gaussdb$#     raise info '%', flag;
gaussdb$#     flag := b IS NULL;
gaussdb$#     raise info '%', flag;
gaussdb$#     flag := c IS NULL;
gaussdb$#     raise info '%', flag;
gaussdb$# end;
gaussdb$# /
INFO:  t
INFO:  t
INFO:  f
ANONYMOUS BLOCK EXECUTE
```

- ^=

  Parameter: nesttable

  Return value: **TRUE** or **FALSE**, of the Boolean type.

  Description: Checks whether two collections are not equal.

  Example:

```
gaussdb=# declare
gaussdb-#     type nest is table of int;
gaussdb-#     a nest := nest(1,2);
gaussdb-#     b nest := nest(1,2);
gaussdb-#     flag bool;
gaussdb-# begin
gaussdb$#     flag := a ^= b;
gaussdb$#     raise info '%', flag;
gaussdb$# end;
gaussdb$# /
INFO:  f
ANONYMOUS BLOCK EXECUTE
```

- [NOT] IN

  Parameter: nesttable

  Return value: **TRUE** or **FALSE**, of the Boolean type.

  Description: Determines whether a collection is in the collection list.

  Example:

```
gaussdb=# declare
gaussdb-#     type tab is table of varchar(10);
gaussdb-#     tmp1 tab := tab('a', 'b');
gaussdb-#     tmp2 tab := tab('a', 'b');
gaussdb-#     flag bool;
gaussdb-# begin
gaussdb$#     flag := tmp2 in (tmp1);
gaussdb$#     raise info '%', flag;
gaussdb$# end;
gaussdb$# /
INFO:  t
ANONYMOUS BLOCK EXECUTE
```

MULTISET

- MULTISET UNION [ALL | DISTINCT]

Parameter: nesttable

Return value: nesttable

Description: Union of two collection variables. The default value **ALL** indicates that duplicate elements are not removed, and **DISTINCT** indicates that duplicate elements are removed.

Example 1: Calculate the union of two collection variables and do not remove duplicate elements. That is, MULTISET UNION ALL.

```
gaussdb=# declare
gaussdb-#     type nest is table of int;
gaussdb-#     a nest := nest(1,2);
gaussdb-#     b nest := nest(2,3);
gaussdb-# begin
gaussdb$#     a := a MULTISET UNION ALL b;
gaussdb$#     raise info '%', a;
gaussdb$# end;
gaussdb$# /
INFO:  {1,2,2,3}
ANONYMOUS BLOCK EXECUTE
```

Example 2: Calculate the union of two collection variables and remove duplicate elements. That is, MULTISET UNION DISTINCT.

```
gaussdb=# declare
gaussdb-#     type nest is table of int;
gaussdb-#     a nest := nest(1,2);
gaussdb-#     b nest := nest(2,3);
gaussdb-# begin
gaussdb$#     a := a MULTISET UNION DISTINCT b;
gaussdb$#     raise info '%', a;
gaussdb$# end;
gaussdb$# /
INFO:  {1,2,3}
ANONYMOUS BLOCK EXECUTE
```

- MULTISET EXCEPT [ALL | DISTINCT]

  Parameter: nesttable

  Return value: nesttable

  Description: Difference of two collection variables. Take A MULTISET EXCEPT B as an example. **ALL** indicates that elements that both A and B have in A are removed and returned. The number of elements to be removed is calculated. For a specific element, if A appears *m* times and B appears *n* times (m > n), the number of elements removed from A is *m* minus *n*. If m ≤ n, *m* elements are removed from A. **DISTINCT** indicates that elements that both A and B have in A are removed and returned. The number of elements to be removed is not calculated. If specific elements appear in A and B, all such elements in A are removed. **ALL** is the default value.

  Example 1: Calculate a difference set of two collection variables, remove elements that both A and B have in A, and return result. The number of elements to be removed is calculated. That is, MULTISET EXCEPT ALL.

```
gaussdb=# declare
gaussdb-#     type nest is table of int;
gaussdb-#     a nest := nest(1,2,2);
gaussdb-#     b nest := nest(2,3);
gaussdb-# begin
gaussdb$#     a := a MULTISET EXCEPT ALL b;
gaussdb$#     raise info '%', a;
gaussdb$# end;
gaussdb$# /
INFO:  {1,2}
ANONYMOUS BLOCK EXECUTE
```


Example 2: Calculate a difference set of two collection variables, remove elements that both A and B have in A, and return result. The number of elements to be removed is not calculated. That is, MULTISET EXCEPT DISTINCT.

```
gaussdb=# declare
gaussdb-#     type nest is table of int;
gaussdb-#     a nest := nest(1,2,2);
gaussdb-#     b nest := nest(2,3);
gaussdb-# begin
gaussdb$#     a := a MULTISET EXCEPT DISTINCT b;
gaussdb$#     raise info '%', a;
gaussdb$# end;
gaussdb$# /
INFO:  {1}
ANONYMOUS BLOCK EXECUTE
```

- MULTISET INTERSECT [ALL | DISTINCT]

  Parameter: nesttable

  Return value: nesttable

  Description: Intersection of two collection variables. Taking A MULTISET INTERSECT B as an example, **ALL** indicates that all duplicate elements in A and B are obtained, and **DISTINCT** indicates that duplicate elements in A and B are obtained and then duplicate elements in this intersection are removed. **ALL** is the default value.

  Example 1: Calculate the intersection of two collection variables and do not remove duplicate elements. That is, MULTISET INTERSECT ALL.

```
gaussdb=# declare
gaussdb-#     type nest is table of int;
gaussdb-#     a nest := nest(1,2,2);
gaussdb-#     b nest := nest(2,2,3);
gaussdb-# begin
gaussdb$#     a := a MULTISET INTERSECT ALL b;
gaussdb$#     raise info '%', a;
gaussdb$# end;
gaussdb$# /
INFO:  {2,2}
ANONYMOUS BLOCK EXECUTE
```

  Example 2: Calculate the intersection of two collection variables and remove duplicate elements. That is, MULTISET INTERSECT DISTINCT.

```
gaussdb=# declare
gaussdb-#     type nest is table of int;
gaussdb-#     a nest := nest(1,2,2);
gaussdb-#     b nest := nest(2,2,3);
gaussdb-# begin
gaussdb$#     a := a MULTISET INTERSECT DISTINCT b;
gaussdb$#     raise info '%', a;
gaussdb$# end;
gaussdb$# /
INFO:  {2}
ANONYMOUS BLOCK EXECUTE
```

Functions of the Collection Type

NOTE

- In the following function definition description, the content in [] is optional. For example, count[()] can be written as count or count().
- In an inner expression, functions of the collection type cannot be called in nesting mode.

- exists(idx)

Parameter: *idx* is of the INT4 or VARCHAR type.

Return value: **TRUE** or **FALSE**, of the Boolean type.

Description: Checks whether a valid element exists in a specified position.

Example:

```
gaussdb=# declare
gaussdb-#     type nest is table of varchar2;
gaussdb-#     a nest := nest('happy','?');
gaussdb-#     flag bool;
gaussdb-# begin
gaussdb$#     flag := a.exists(1);
gaussdb$#     raise info '%', flag;
gaussdb$#     flag := a.exists(10);
gaussdb$#     raise info '%', flag;
gaussdb$# end;
gaussdb$# /
INFO:  t
INFO:  f
ANONYMOUS BLOCK EXECUTE

gaussdb=# declare
gaussdb-#     type nest is table of varchar2 index by varchar2;
gaussdb-#     a nest;
gaussdb-#     flag bool;
gaussdb-# begin
gaussdb$#     a('1') := 'Be';
gaussdb$#     a('2') := 'happy';
gaussdb$#     a('3') := '.';
gaussdb$#     flag := a.exists('1');
gaussdb$#     raise info '%', flag;
gaussdb$#     flag := a.exists('ddd');
gaussdb$#     raise info '%', flag;
gaussdb$# end;
gaussdb$# /
INFO:  t
INFO:  f
ANONYMOUS BLOCK EXECUTE
```

- extend[(count[, idx])]

  Parameters: *idx* and *count* are of the INT4 type.

  Return value: No value is returned.

  Description: Supports only variables of the nest-table type. One or *count* elements are extended at the end of the nest-table variable. If index set element *idx* exists, *count* index elements are copied to the end of the variable.

  Example 1: A nest-table variable extends 1 element and the extended element value is **NULL**.

  ```
  gaussdb=# declare
  gaussdb-#     type nest is table of int;
  gaussdb-#     a nest := nest(1);
  gaussdb-# begin
  gaussdb$#     raise info '%', a;
  gaussdb$#     a.extend;
  gaussdb$#     raise info '%', a;
  gaussdb$#     a.extend;
  gaussdb$#     raise info '%', a;
  gaussdb$# end;
  gaussdb$# /
  INFO:  {1}
  INFO:  {1,NULL}
  INFO:  {1,NULL,NULL}
  ANONYMOUS BLOCK EXECUTE
  ```

  Example 2: A nest-table variable extends *n* elements and the extended element value is **NULL**.


```
gaussdb=# declare
gaussdb-#     type nest is table of int;
gaussdb-#     a nest := nest(1);
gaussdb-# begin
gaussdb$#     raise info '%', a;
gaussdb$#     a.extend(2);
gaussdb$#     raise info '%', a;
gaussdb$# end;
gaussdb$# /
INFO:  {1}
INFO:  {1,NULL,NULL}
ANONYMOUS BLOCK EXECUTE
```

Example 3: A nest-table variable extends *n* elements and the extended element value is the value of the specified index element.

```
gaussdb=# declare
gaussdb-#     type nest is table of int;
gaussdb-#     a nest := nest(9);
gaussdb-# begin
gaussdb$#     raise info '%', a;
gaussdb$#     a.extend(2,1);
gaussdb$#     raise info '%', a;
gaussdb$# end;
gaussdb$# /
INFO:  {9}
INFO:  {9,9,9}
ANONYMOUS BLOCK EXECUTE
```

- delete[(idx1[, idx2])]

  Parameter: *idx1* and *idx2* are of the int4 or varchar2 type.

  Return value: No value is returned.

  Description: If there is no parameter, the variable of the nest-table type deletes all elements of the collection type and the space. If the space is used later, the space needs to be expanded. If there is no parameter, the variable of the index-by table type deletes all elements. If there is one parameter, the element in the specified position is deleted (the space is not deleted). If there are two parameters, all elements in the index interval are deleted (the space is not deleted).

  Example 1: Delete all elements and spaces of the collection type from the variable of the nest-table collection type.

```
gaussdb=# declare
gaussdb-#     type nest is table of int;
gaussdb-#     a nest := nest(1,2,3,4,5);
gaussdb-# begin
gaussdb$#     raise info '%', a;
gaussdb$#     a.delete();
gaussdb$#     raise info '%', a;
gaussdb$# end;
gaussdb$# /
INFO:  {1,2,3,4,5}
INFO:  {}
ANONYMOUS BLOCK EXECUTE
```

  Example 2: Delete an element at a specified position from the variable of the nest-table collection type.

```
gaussdb=# declare
gaussdb-#     type nest is table of int;
gaussdb-#     a nest := nest(1,2,3,4,5);
gaussdb-# begin
gaussdb$#     raise info '%', a;
gaussdb$#     a.delete(3);
gaussdb$#     raise info '%', a;
gaussdb$#     a(3) := 3;
gaussdb$#     raise info '%', a;
```



```
gaussdb$# end;
gaussdb$# /
INFO:  {1,2,3,4,5}
INFO:  {1,2,4,5}
INFO:  {1,2,3,4,5}
ANONYMOUS BLOCK EXECUTE
```

Example 3: Delete elements in a specified interval from the variable of the nest-table collection type.

```
gaussdb=# declare
gaussdb-#     type nest is table of int;
gaussdb-#     a nest := nest(1,2,3,4,5);
gaussdb-# begin
gaussdb$#     raise info '%', a;
gaussdb$#     a.delete(2,4);
gaussdb$#     raise info '%', a(1);
gaussdb$#     raise info '%', a(5);
gaussdb$#     raise info '%', a;
gaussdb$# end;
gaussdb$# /
INFO:  {1,2,3,4,5}
INFO:  1
INFO:  5
INFO:  {1,5}
ANONYMOUS BLOCK EXECUTE
```

Example 4: Delete all elements and spaces of the collection type from the variable of the index-by table collection type.

```
gaussdb=# declare
gaussdb-#     type t1 is table of int index by varchar;
gaussdb-#     v t1 := t1('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4);
gaussdb-# begin
gaussdb$#     v.delete();
gaussdb$#     raise info '%', v.count();
gaussdb$# end;
gaussdb$# /
INFO:  0
ANONYMOUS BLOCK EXECUTE
```

Example 5: Delete an element at a specified position from the variable of the index-by table collection type.

```
gaussdb=# declare
gaussdb-#     type t1 is table of int index by varchar;
gaussdb-#     v t1 := t1('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4);
gaussdb-# begin
gaussdb$#     raise info '%', v('a');
gaussdb$#     v.delete('a');
gaussdb$#     raise info '%', v('a');
gaussdb$# end;
gaussdb$# /
INFO:  1
ERROR:  no data found
CONTEXT:  PL/pgSQL function inline_code_block line 6 at RAISE
```

Example 6: Delete elements in a specified interval from the variable of the index-by table collection type.

```
gaussdb=# declare
gaussdb-#     type t1 is table of int index by varchar;
gaussdb-#     v t1 := t1('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4);
gaussdb-# begin
gaussdb$#     raise info '%', v('b');
gaussdb$#     v.delete('a', 'c');
gaussdb$#     raise info '%', v('b');
gaussdb$# end;
gaussdb$# /
INFO:  2
ERROR:  no data found
CONTEXT:  PL/pgSQL function inline_code_block line 6 at RAISE
```


- trim[(n)]

  Parameter: *n* is of the INT4 type.

  Return value: No value is returned.

  Description: Supports only variables of the nest-table type. If no parameter is specified, the last element space is deleted. If the input parameter is valid, the specified number of element spaces are deleted.

  Example:

```
gaussdb=# declare
gaussdb-#     type nest is table of int;
gaussdb-#     aa nest := nest(11,22,33,44,55);
gaussdb-# begin
gaussdb$#     raise info 'aa:%' ,aa;
gaussdb$#     aa.trim;  -- No parameter
gaussdb$#     raise info 'aa:%' ,aa;
gaussdb$#     aa.trim();  -- No parameter
gaussdb$#     raise info 'aa:%' ,aa;
gaussdb$#     aa.trim(2);  -- Valid parameter
gaussdb$#     raise info 'aa:%' ,aa;
gaussdb$#     aa.trim(2);  -- The collection element space is less than 2. An error is reported when the
parameter is invalid.
gaussdb$# end;
gaussdb$# /
INFO:  aa:{11,22,33,44,55}
INFO:  aa:{11,22,33,44}
INFO:  aa:{11,22,33}
INFO:  aa:{11}
ERROR:  Subscript beyond count
CONTEXT:  PL/pgSQL function inline_code_block line 11 at assignment
```

- count[()]

  Parameter: none

  Return value: INT type

  Description: Returns the number of valid elements in a collection.

  Example 1: Use the count function for the variable of the nest-table collection type.

```
gaussdb=# declare
gaussdb-# type nest is table of int;
gaussdb-# aa nest:=nest(11,22,33,44,55);
gaussdb-# begin
gaussdb$# raise info 'count:%' ,aa.count;
gaussdb$# aa.delete(3);  -- Delete an element. The element whose index is 3 is invalid.
gaussdb$# raise info 'count:%' ,aa.count();
gaussdb$# end;
gaussdb$# /
INFO:  count:5
INFO:  count:4
ANONYMOUS BLOCK EXECUTE
```

  Example 2: Use the count function for the variable of the index-by table collection type.

```
gaussdb=# declare
gaussdb-# type t1 is table of int index by int;
gaussdb-# aa t1;
gaussdb-# begin
gaussdb$# aa(1) := 111;
gaussdb$# aa(2) := 222;
gaussdb$# aa(3) := 333;
gaussdb$# raise info 'count:%' ,aa.count();
gaussdb$# end;
gaussdb$# /
INFO:  count:3
ANONYMOUS BLOCK EXECUTE
```



```
gaussdb=# declare
gaussdb-# type t1 is table of int index by varchar;
gaussdb-# aa t1;
gaussdb-# begin
gaussdb$# aa('aaa') := 111;
gaussdb$# aa('bbb') := 222;
gaussdb$# aa('ccc') := 333;
gaussdb$# raise info 'count:%' ,aa.count;
gaussdb$# end;
gaussdb$# /
INFO:  count:3
ANONYMOUS BLOCK EXECUTE
```

- first[()]

  Parameter: none

  Return value: INT or VARCHAR

  Description: Returns the index of the first valid element in a collection.

  Example 1: Use the first function for the variable of the nest-table collection type.

```
gaussdb=# declare
gaussdb-# type nest is table of int;
gaussdb-# aa nest:=nest(11,22,33,44,55);
gaussdb-# begin
gaussdb$# raise info 'first:%' ,aa.first();
gaussdb$# aa.delete(1);
gaussdb$# raise info 'first:%' ,aa.first;  -- The element whose index is 1 is invalid. The index of the first valid element is 2.
gaussdb$# end;
gaussdb$# /
INFO:  first:1
INFO:  first:2
ANONYMOUS BLOCK EXECUTE
```

  Example 2: Use the first function for the variable of the index-by table collection type.

```
gaussdb=# declare
gaussdb-# type t1 is table of int index by int;
gaussdb-# aa t1;
gaussdb-# begin
gaussdb$# aa(3) := 111;
gaussdb$# aa(2) := 222;
gaussdb$# aa(1) := 333;
gaussdb$# raise info 'first:%' ,aa.first;
gaussdb$# end;
gaussdb$# /
INFO:  first:1
ANONYMOUS BLOCK EXECUTE

gaussdb=# declare
gaussdb-# type t1 is table of int index by varchar;
gaussdb-# aa t1;
gaussdb-# begin
gaussdb$# aa('aaa') := 111;
gaussdb$# aa('bbb') := 222;
gaussdb$# aa('ccc') := 333;
gaussdb$# raise info 'first:%' ,aa.first;
gaussdb$# end;
gaussdb$# /
INFO:  first:aaa
ANONYMOUS BLOCK EXECUTE
```

- last[()]

  Parameter: none

  Return value: INT or VARCHAR

Description: Returns the index of the last valid element in a collection.

Example 1: Use the last function for the variable of the nest-table collection type.

```
gaussdb=# declare
gaussdb-# type nest is table of int;
gaussdb-# aa nest:=nest(11,22,33,44,55);
gaussdb-# begin
gaussdb$# raise info 'last:%' ,aa.last;
gaussdb$# aa.delete(5);
gaussdb$# raise info 'last:%' ,aa.last();  -- The element whose index is 5 is invalid. The index of the last valid element is 4.
gaussdb$# end;
gaussdb$# /
INFO:  last:5
INFO:  last:4
ANONYMOUS BLOCK EXECUTE
```

Example 2: Use the last function for the variable of the index-by table collection type.

```
gaussdb=# declare
gaussdb-# type t1 is table of int index by varchar;
gaussdb-# aa t1;
gaussdb-# begin
gaussdb$# aa(3) := 111;
gaussdb$# aa(2) := 222;
gaussdb$# aa(1) := 333;
gaussdb$# raise info 'last:%' ,aa.last();
gaussdb$# end;
gaussdb$# /
INFO:  last:3
ANONYMOUS BLOCK EXECUTE

gaussdb=# declare
gaussdb-# type t1 is table of int index by varchar;
gaussdb-# aa t1;
gaussdb-# begin
gaussdb$# aa('aaa') := 111;
gaussdb$# aa('bbb') := 222;
gaussdb$# aa('ccc') := 333;
gaussdb$# raise info 'last:%' ,aa.last;
gaussdb$# end;
gaussdb$# /
INFO:  last:ccc
ANONYMOUS BLOCK EXECUTE
```

- prior(idx)

  Parameter: *idx* is of the INT or VARCHAR type.

  Return value: int or varchar type

  Description: Returns the index of a valid element before the given index in a collection.

  Example 1: Use the prior function for the variable of the nest-table collection type.

  ```
  gaussdb=# declare
  gaussdb-# type nest is table of int;
  gaussdb-# aa nest:=nest(11,22,33,44,55);
  gaussdb-# begin
  gaussdb$# raise info 'prior:%' ,aa.prior(3);
  gaussdb$# end;
  gaussdb$# /
  INFO:  prior:2
  ANONYMOUS BLOCK EXECUTE
  ```

  Example 2: Use the prior function for the variable of the index-by table collection type.


```
gaussdb=# declare
gaussdb-# type t1 is table of int index by varchar;
gaussdb-# aa t1;
gaussdb-# begin
gaussdb$# aa('ccc') := 111;
gaussdb$# aa('bbb') := 222;
gaussdb$# aa('aaa') := 333;
gaussdb$# raise info 'prior:%' ,aa.prior('bbb');
gaussdb$# raise info 'prior:%' ,aa.prior('ddd');
gaussdb$# end;
gaussdb$# /
INFO:  prior:aaa
INFO:  prior:ccc
ANONYMOUS BLOCK EXECUTE
```

- next(idx)

  Parameter: *idx* is of the INT or VARCHAR type.

  Return value: INT or VARCHAR

  Description: Returns the index of a valid element following the given index in a collection.

  Example 1: Use the next function for the variable of the nest-table collection type.

```
gaussdb=# declare
gaussdb-# type nest is table of int;
gaussdb-# aa nest:=nest(11,22,33,44,55);
gaussdb-# begin
gaussdb$# raise info 'next:%' ,aa.next(3);
gaussdb$# end;
gaussdb$# /
INFO:  next:4
ANONYMOUS BLOCK EXECUTE
```

  Example 2: Use the next function for the variable of the index-by table collection type.

```
gaussdb=# declare
gaussdb-# type t1 is table of int index by int;
gaussdb-# aa t1;
gaussdb-# begin
gaussdb$# aa(3) := 111;
gaussdb$# aa(2) := 222;
gaussdb$# aa(1) := 333;
gaussdb$# raise info 'next:%' ,aa.next(2);
gaussdb$# raise info 'next:%' ,aa.next(-999);
gaussdb$# end;
gaussdb$# /
INFO:  next:3
INFO:  next:1
ANONYMOUS BLOCK EXECUTE
```

- limit

  Parameter: none

  Return value: null

  Description: Returns null values and only for variables of the nest-table type.

  Example:

```
gaussdb=# declare
gaussdb-# type nest is table of int;
gaussdb-# aa nest:=nest(11,22,33,44,55);
gaussdb-# begin
gaussdb$# raise info 'limit:%' ,aa.limit;
gaussdb$# end;
gaussdb$# /
INFO:  limit:<NULL>
ANONYMOUS BLOCK EXECUTE
```


Collection-related Functions

- unnest_table (anynesttable) or unnest (anynesttable)

  Parameter: any nest-table collection type.

  Description: Returns the result set of all valid elements in a given nest-table. Multiple rows of data are returned.

  Return type: setof anyelement.

  Example:

```
gaussdb=# create or replace procedure p1()
gaussdb-# as
gaussdb$#     type t1 is table of int;
gaussdb$#     v2 t1 := t1(null, 2, 3, 4, null);
gaussdb$#     tmp int;
gaussdb$#     cursor c1 is select * from unnest_table(v2);
gaussdb$# begin
gaussdb$# open c1;
gaussdb$# for i in 1 .. v2.count loop
gaussdb$#     fetch c1 into tmp;
gaussdb$#     if tmp is null then
gaussdb$#         dbe_output.print_line(i || ': is null');
gaussdb$#     else
gaussdb$#         dbe_output.print_line(i || ': ' || tmp);
gaussdb$#     end if;
gaussdb$# end loop;
gaussdb$# close c1;
gaussdb$# end;
gaussdb$# /
CREATE PROCEDURE
gaussdb=# call p1();
1: is null
2: 2
3: 3
4: 4
5: is null
 p1
----

(1 row)
gaussdb=# drop procedure if exists p1();
DROP PROCEDURE

-- Example: nested record types in a nest-table
gaussdb=# create or replace procedure p1() is
gaussdb$#   type rec is record(c1 int, c2 int);
gaussdb$#   type t1 is table of rec;
gaussdb$#   v t1 := t1(rec(1, 1), rec(2, null), rec(null, null), null);
gaussdb$#   v2 t1 := t1();
gaussdb$#   cursor cur is select * from unnest(v);
gaussdb$# begin
gaussdb$#   v2.extend(v.count);
gaussdb$#   open cur;
gaussdb$#   for i in 1 .. v.count loop
gaussdb$#     fetch cur into v2(i);
gaussdb$#     raise info '%', v2(i);
gaussdb$#   end loop;
gaussdb$#   close cur;
gaussdb$# end;
gaussdb$# /
CREATE PROCEDURE
gaussdb=# call p1();
INFO:  (1,1)
INFO:  (2,)
INFO:  (,)
INFO:  (,)
 p1
----
```



```
(1 row)
gaussdb=# drop procedure if exists p1();
DROP PROCEDURE
```

NOTE

If the element type of the collection is record and any element is NULL, the element is not returned. Instead, a non-null record value is returned. All columns of the record are NULL. For details, see the example.

- unnest_table (anyindexbytable) or unnest (anyindexbytable)

  Parameter: any index-by table collection type.

  Description: Returns the result set of all elements sorted by index in a given index-by table. Multiple rows of data are returned.

  Return type: setof anyelement

  Constraint: Only the index-by int type is supported. The index-by varchar type is not supported.

  Example:

```
gaussdb=# create or replace procedure p1()
gaussdb-# as
gaussdb$#     type t1 is table of int index by int;
gaussdb$#     v2 t1 := t1(1=>1, -10=>(-10), 6=>6, 4=>null);
gaussdb$#     tmp int;
gaussdb$#     cursor c1 is select * from unnest_table(v2);
gaussdb$# begin
gaussdb$# open c1;
gaussdb$# for i in 1 .. v2.count loop
gaussdb$#     fetch c1 into tmp;
gaussdb$#     if tmp is null then
gaussdb$#         dbe_output.print_line(i || ': is null');
gaussdb$#     else
gaussdb$#         dbe_output.print_line(i || ': ' || tmp);
gaussdb$#     end if;
gaussdb$# end loop;
gaussdb$# close c1;
gaussdb$# end;
gaussdb$# /
CREATE PROCEDURE
gaussdb=# call p1();
1: -10
2: 1
3: is null
4: 6
 p1
----

(1 row)
gaussdb=# drop procedure if exists p1();
DROP PROCEDURE

-- Example: nested record types in an index-by table
gaussdb=# create or replace procedure p1() is
gaussdb$#   type rec is record(c1 int, c2 int);
gaussdb$#   type t1 is table of rec index by int;
gaussdb$#   v t1 := t1(1 => rec(1, 1), 2 => rec(2, null), 3 => rec(null, null), 4 => null);
gaussdb$#   v2 t1 := t1();
gaussdb$#   cursor cur is select * from unnest(v);
gaussdb$# begin
gaussdb$#   open cur;
gaussdb$#   for i in 1 .. v.count loop
gaussdb$#     fetch cur into v2(i);
gaussdb$#     raise info '%', v2(i);
gaussdb$#   end loop;
gaussdb$#   close cur;
```



```
gaussdb$# end;
gaussdb$# /
CREATE PROCEDURE
gaussdb=# call p1();
INFO:  (1,1)
INFO:  (2,)
INFO:  (,)
INFO:  (,)
 p1
----

(1 row)
gaussdb=# drop procedure if exists p1();
DROP PROCEDURE
```

NOTE

If the element type of the collection is record and any element is NULL, the element is not returned. Instead, a non-null record value is returned. All columns of the record are NULL. For details, see the example.

10.4.3 Records

Record Variables

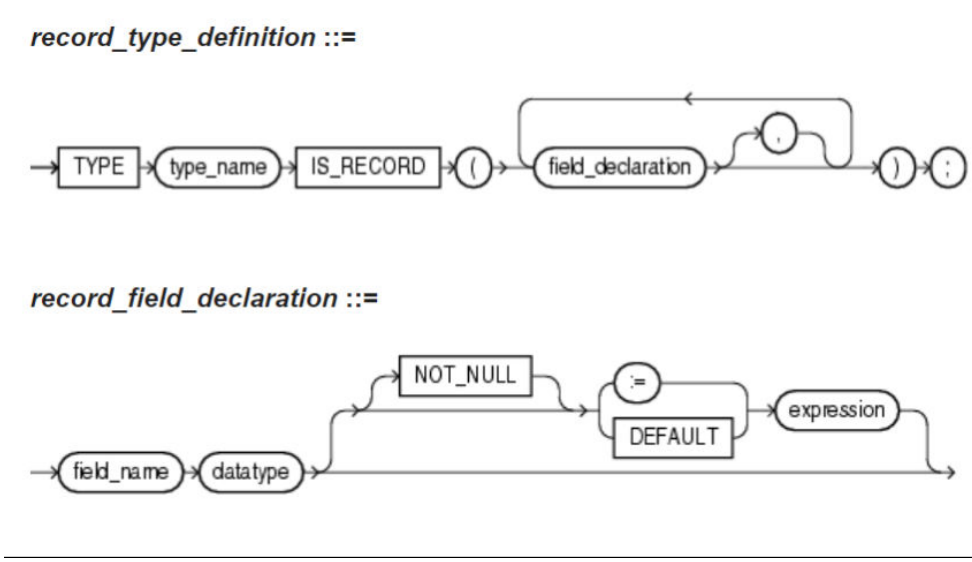
Perform the following operations to create a record variable:

Define a record type and use this type to declare a variable.

Syntax

For the syntax of the record type, see **Figure 10-1**.

Figure 10-1 Syntax of the record type



The above syntax diagram is explained as follows:

- **record_type**: record type
- **field**: record columns

- **datatype**: record data type
- **expression**: expression for setting a default value

NOTE

In GaussDB:

- When assigning values to record variables, you can:
  - Declare a record type and define member variables of this type when you declare a function or stored procedure.
  - Assign the value of a record variable to another record variable.
  - Use **SELECT INTO** or **FETCH** to assign values to a record type.
  - Assign the **NULL** value to a record variable.
- The record type constructor can use => to assign values. The restrictions are as follows:
  - The => function is used when the record type constructor assigns values. This function is supported only when the A-compatible database is used (**sql_compatibility** is set to **'A'**).
  - The record type can be defined in a package.
  - The record type constructor uses the => function to assign values. This function is supported only when => is consecutively used to assign values to input parameters, for example, **record_name(elename1 => val1, elename2 => val2, elename3 => val3)** or **record_name(val1, elename2 => val2, elename3 => val3)**. Discontinuous use of => or => is not supported. For example, **record_name(elename1 => val1, elename2 => val2, val3)** or **record_name(val1, elename2 => val2, val3)**, where the last input parameter is not assigned a value.
- The **INSERT** and **UPDATE** statements cannot use a record variable to insert or update data.
- The constructor of the record type cannot be used as the default value of a function or stored procedure parameter.
- Just like a variable, a record column of the compound type does not have a default value in the declaration.
- If a member has a record type, the default value of the inner record type cannot be not transferred to the outer record type.
- If **set enable_recordtype_check_strict** is set to **on** and the member is of the record type and a column of the record type has the not null or default attribute, an error is reported during stored procedure or package compilation.
- If **set enable_recordtype_check_strict** is set to **on**, a package contains the record type, and a column of the record type has the not null or default attribute, an error is reported during creation or compilation.
- If **set enable_recordtype_check_strict** is set to **on**, the not null function of the record type in the stored procedure takes effect.
- The data type can be the record type, array type, or collection type defined in a stored procedure (anonymous blocks are not supported).
- After the GUC parameter **set behavior_compat_options** is set to **'proc_outparam_override'**:
  - Functions that contain the **out** output parameter and returns data of the record type can be invoked using SELECT and CALL in external SQL statements, and can be invoked using PERFORM and expressions in stored procedures.
  - If the function returns data of the undefined record type, at least one **out** parameter is required. If the function returns data of the defined record type, the **out** parameter is optional. For details, see the example.
  - Functions that contain the **out** output parameter and return data of the composite or record type, ensure that the expected return type of the function is the same as the actual return type.

Examples

```
-- Create a table and insert some data into the table.
gaussdb=# create table emp_rec(
gaussdb(#     empno numeric(4,0) not null,
gaussdb(#     ename varchar(10)
gaussdb(# );
CREATE TABLE
gaussdb=# insert into emp_rec values(111, 'aaa'), (222, 'bbb'), (333, 'ccc');
INSERT 0 3
-- The table is defined as follows:
gaussdb=# \d emp_rec
          Table "public.emp_rec"
 Column |         Type          | Modifiers
--------+-----------------------+-----------
 empno  | numeric(4,0)          | not null
 ename  | character varying(10) |

-- Perform record operations in the function.
gaussdb=# CREATE OR REPLACE FUNCTION regress_record(p_w VARCHAR2) RETURNS VARCHAR2 AS $$
gaussdb$# DECLARE
gaussdb$#     -- Declare a record type.
gaussdb$#     type rec_type is record (name  varchar2(100), epno int);
gaussdb$#     employer rec_type;
gaussdb$#     -- Use %type to declare the record type.
gaussdb$#     type rec_type1 is record (name  emp_rec.ename%type, epno int :=10);
gaussdb$#     employer1 rec_type1;
gaussdb$#     -- Declare a record type with the default value.
gaussdb$#     type rec_type2 is record (
gaussdb$#         name varchar2 not null := 'SCOTT',
gaussdb$#         epno int not null :=10
gaussdb$#     );
gaussdb$#     employer2 rec_type2;
gaussdb$#     CURSOR C1 IS  select ename,empno from emp_rec order by 1 limit 1;
gaussdb$# BEGIN
gaussdb$#     -- Assign a value to a member record variable.
gaussdb$#     employer.name := 'WARD';
gaussdb$#     employer.epno = 18;
gaussdb$#     raise info 'employer name: % , epno:%', employer.name, employer.epno;
gaussdb$#
gaussdb$#     -- Assign the value of a record variable to another variable.
gaussdb$#     employer1 := employer;
gaussdb$#     raise info 'employer1 name: % , epno: %',employer1.name, employer1.epno;
gaussdb$#
gaussdb$#     -- Assign the NULL value to a record variable.
gaussdb$#     employer1 := NULL;
gaussdb$#     raise info 'employer1 name: % , epno: %',employer1.name, employer1.epno;
gaussdb$#
gaussdb$#     -- Obtain the default value of a record variable.
gaussdb$#     raise info 'employer2 name: % ,epno: %', employer2.name, employer2.epno;
gaussdb$#
gaussdb$#     -- Use a record variable in the FOR loop.
gaussdb$#     for employer in select ename,empno from emp_rec order by 1 limit 1 loop
gaussdb$#         raise info 'employer name: % , epno: %', employer.name, employer.epno;
gaussdb$#     end loop;
gaussdb$#
gaussdb$#     -- Use a record variable in the SELECT INTO statement.
gaussdb$#     select ename,empno  into employer2 from emp_rec order by 1 limit 1;
gaussdb$#     raise info 'employer name: % , epno: %', employer2.name, employer2.epno;
gaussdb$#
gaussdb$#     -- Use a record variable in a cursor.
gaussdb$#     OPEN C1;
gaussdb$#     FETCH C1 INTO employer2;
gaussdb$#     raise info 'employer name: % , epno: %', employer2.name, employer2.epno;
gaussdb$#     CLOSE C1;
gaussdb$#     RETURN employer.name;
gaussdb$# END; $$
gaussdb-# LANGUAGE plpgsql;
CREATE FUNCTION
```



```
-- Call this function.
gaussdb=# CALL regress_record('abc');
INFO:  employer name: WARD , epno:18
INFO:  employer1 name: WARD , epno: 18
INFO:  employer1 name: <NULL> , epno: <NULL>
INFO:  employer2 name: SCOTT ,epno: 10
INFO:  employer name: aaa , epno: 111
INFO:  employer name: aaa , epno: 111
INFO:  employer name: aaa , epno: 111
 regress_record
----------------
 aaa
(1 row)
-- Delete the function.
gaussdb=# DROP FUNCTION regress_record;
DROP FUNCTION
-- Delete the table.
gaussdb=# DROP TABLE emp_rec;
DROP TABLE

-- If the compatibility parameter proc_outparam_override is enabled, the data of the defined record type is returned. The out parameter is not required for the function.
gaussdb=# create type rec_type is (c1 int, c2 int);
CREATE TYPE
gaussdb=# set behavior_compat_options = 'proc_outparam_override';
SET
gaussdb=# create or replace function func(a in int) return rec_type is
gaussdb$#     r rec_type;
gaussdb$# begin
gaussdb$#     r.c1:=1;
gaussdb$#     r.c2:=1;
gaussdb$#     return r;
gaussdb$# end;
gaussdb$# /
CREATE FUNCTION
gaussdb=# call func(0);
 c1 | c2
----+----
  1 |  1
(1 row)
gaussdb=# drop function func;
DROP FUNCTION
gaussdb=# drop type rec_type;
DROP TYPE
-- If the compatibility parameter proc_outparam_override is enabled, the data of the undefined record type is returned and at least one out parameter is required for the function
gaussdb=# set behavior_compat_options = 'proc_outparam_override';
SET
gaussdb=# create or replace function func(a out int) return record is
gaussdb$#     type rc is record(c1 int, c2 int);
gaussdb$#     r rc;
gaussdb$# begin
gaussdb$#     r.c1 := 1;
gaussdb$#     r.c2 := 1;
gaussdb$#     a := 999;
gaussdb$#     return r;
gaussdb$# end;
gaussdb$# /
CREATE FUNCTION
gaussdb=# call func(1);
 func  |  a
-------+-----
 (1,1) | 999
(1 row)
gaussdb=# drop function func;
DROP FUNCTION

-- Create an A-compatible database and switch to this database.
gaussdb=#  CREATE DATABASE ora_compatible_db DBCOMPATIBILITY 'A';
```



```
CREATE DATABASE
gaussdb=#  \c ora_compatible_db;
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "ora_compatible_db" as user "omm".
-- Define the record in the pkg. Use => to assign values to all input parameters of the record constructor.
ora_compatible_db=#
CREATE or REPLACE package PA_TEST1
AS
 PROCEDURE P_DISPLAY1(a NUMBER, b  VARCHAR);
 TYPE t1 IS record(va int,vb int);
 v1 t1;
END PA_TEST1;
/
ora_compatible_db=# CREATE or REPLACE package body PA_TEST1
AS
  PROCEDURE P_DISPLAY1(a NUMBER, b  VARCHAR) AS
 BEGIN
  v1 := t1(va => 1, vb => 2);
  dbe_output.put_line('pkg test1');
 END;
END PA_TEST1;
/

ora_compatible_db=#  CALL PA_TEST1.P_DISPLAY1(b => 'm', a => 1);
 p_display1
------------

(1 row)
-- Define the record in the pkg. The input parameter values of the record constructor are not consecutively
used =>, and an error message is displayed.
ora_compatible_db=#
CREATE or REPLACE package PA_TEST1
AS
 PROCEDURE P_DISPLAY1(a NUMBER, b  VARCHAR);
 TYPE t1 IS record(va int,vb int);
 v1 t1;
END PA_TEST1;
/
CREATE or REPLACE package body PA_TEST1
AS
  PROCEDURE P_DISPLAY1(a NUMBER, b  VARCHAR) AS
 BEGIN
  v1 := t1(va => 1, 2);
  dbe_output.put_line('pkg test1');
 END;
END PA_TEST1;
/
ora_compatible_db=# CALL PA_TEST1.P_DISPLAY1(b => 'm', a => 1);
ERROR:  positional argument cannot follow named argument
LINE 1: SELECT "17347.t1"(va => 1, 2)
                                ^
QUERY:  SELECT "17347.t1"(va => 1, 2)
CONTEXT:  referenced column: 17347.t1
PL/pgSQL function p_display1(numeric,character varying) line 2 at assignment
-- Delete the created database and switch back to the original database.
ora_compatible_db=# \c postgres;
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "postgres" as user "omm".
gaussdb=# DROP DATABASE ora_compatible_db;
DROP DATABASE
```

10.5 DECLARE Syntax

10.5.1 Basic Structure

Structure

A PL/SQL block can contain a sub-block which can be placed in any section. The following describes the architecture of a PL/SQL block:

- **DECLARE**: declares variables, types, cursors, and regional stored procedures and functions used in the PL/SQL block.
  ```
  DECLARE
  ```

  NOTE

  This part is optional if no variable needs to be declared.
  - An anonymous block may omit the DECLARE keyword if no variable needs to be declared.
  - For a stored procedure, AS is used, which is equivalent to DECLARE. The AS keyword must be reserved even if there is no variable declaration section.
- **EXECUTION**: specifies procedure and SQL statements. It is the main part of a program. Mandatory.
  ```
  BEGIN
  ```
- Exception part: processes errors. Optional.
  ```
  EXCEPTION
  ```
- End. Mandatory.
  ```
  END;
  /
  ```

NOTICE

You are not allowed to use consecutive tabs in the PL/SQL block because they may result in an exception when the gsql tool is executed with the **-r** parameter specified.

Category

PL/SQL blocks are classified into the following types:

- Anonymous block: a dynamic block that can be executed only for once. For details about the syntax, see **Figure 10-2**.
- Subprogram: a stored procedure, function, operator, or packages stored in a database. A subprogram created in a database can be called by other programs.

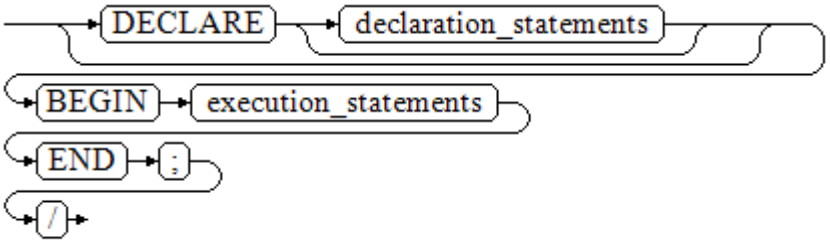
10.5.2 Anonymous Blocks

An anonymous block applies to a script infrequently executed or a one-off activity. An anonymous block is executed in a session and is not stored.

Syntax

Figure 10-2 shows the syntax diagrams for an anonymous block.

Figure 10-2 anonymous_block::=



Details about the syntax diagram are as follows:

- The execute part of an anonymous block starts with a **BEGIN** statement, has a break with an **END** statement, and ends with a semicolon (;). Type a slash (/) and press **Enter** to execute the statement.

  NOTICE

  The terminator "/" must be written in an independent row.

- The declaration section includes the variable definition, type, and cursor definition.
- A simplest anonymous block does not execute any commands. At least one statement, even a NULL statement, must be presented in any implementation blocks.

The following lists basic anonymous block programs:

```
-- Null statement block
gaussdb=# BEGIN
    NULL;
END;
/
ANONYMOUS BLOCK EXECUTE
-- Display information on the console.
gaussdb=# BEGIN
    dbe_output.print_line('hello world!');
END;
/
hello world!
ANONYMOUS BLOCK EXECUTE
-- Display variable content on the console.
gaussdb=# DECLARE
    my_var VARCHAR2(30);
BEGIN
    my_var :='world';
    dbe_output.print_line('hello'||my_var);
END;
/
helloworld
ANONYMOUS BLOCK EXECUTE
```

10.5.3 Subprogram

A subprogram stores stored procedures, functions, operators, and advanced packages. A subprogram created in a database can be called by other programs.

10.5.3.1 Standalone Subprograms

Subprograms created in a schema include stored procedures, functions, and packages created in a schema.

For details, see:

- **CREATE PACKAGE**
- **CREATE FUNCTION**
- **CREATE PROCEDURE**

10.5.3.2 Package Subprograms

Subprograms created in a package include stored procedures and functions created and declared in a package.

For details, see **CREATE PACKAGE**.

10.5.3.3 Nested Subprograms

Subprograms created in a PL/SQL block include sub-stored-procedures or subfunctions declared and created in anonymous blocks, stored procedures, functions, and stored procedures and functions in packages.

Precautions

- This is used in the A compatibility database.
- The maximum number of nesting layers is specified by the **max_subpro_nested_layers** parameter. The default value is **3**, and the value range is 0 to 100. If a nested subprogram contains an anonymous block, the layer of anonymous block is not counted, but the nested subprograms in the anonymous block are counted in the total number of layers.
- Nested subprograms do not support reloading or SETOF.
- Nested subprograms cannot be defined as autonomous transactions. They can call stored procedures or functions that contain autonomous transactions.
- Subfunctions (FUNCTION) cannot be directly called and must have return values. Sub-stored-procedures (PROCEDURE) cannot be called in expressions.
- Nested subprograms cannot be called by PERFORM. Dynamic statements cannot contain nested subprograms.
- Currently, the following modifiers are supported for nested subprograms. Other modifiers are not supported.
  ```
  {IMMUTABLE  | STABLE  | VOLATILE }
  {CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
  ```
- Only one qualifier can reference a nested subprogram or a variable of a nested subprogram.
- When the return value type of a subfunction (FUNCTION) is the record type customized by the function, subfunc().col cannot be used to access the column attribute of the return value of the subfunction. As a result, an error is reported during execution.
- The declaration of a nested subprogram must be at the end of the declaration part (declare the nested subprogram after the declaration of variables, cursors, and types is complete).

- Nested subprograms can be called only inside declared functions or stored procedures and cannot be used externally.
- The debugger breakpoint is not supported when nested subprograms are used. Step-by-step debugging is supported.
- Other precautions are the same as those for stored procedures and functions.

Syntax

- Syntax format for creating a sub-stored-procedure:

```
PROCEDURE procedure_name [ (parameters) ]
    [{IMMUTABLE  | STABLE  | VOLATILE }
     | {CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }]
    { IS | AS }
    [ declarations ]
BEGIN
    plsql_body
END;
```

- Syntax for creating a subfunction:

```
FUNCTION function_name  [ (parameters) ] RETURN rettype
    [{IMMUTABLE  | STABLE  | VOLATILE }
     | {CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }]
    { IS | AS }
    [ declarations ]
BEGIN
    plsql_body
END;
```

  In the **declarations** part, you can define the nested subprograms of the lower layer.

- Example:

```
-- Create a stored procedure.
CREATE OR REPLACE PROCEDURE proc_test() AS
    -- Declare and define a sub-stored-procedure.
    PROCEDURE proc_sub() IS
    BEGIN
        dbe_output.put_line('this is subpragram');
    END;
BEGIN
    dbe_output.put_line('this is a procedure');
    -- Call a sub-stored-procedure in an execution block.
    proc_sub();
END;
/
    -- Call a stored procedure externally.
BEGIN
    proc_test;
END;
/
-- Output the result.
this is a procedure
this is subpragram
ANONYMOUS BLOCK EXECUTE
```

Declaration and Definition Rules of Nested Subprograms

- Nested subprograms cannot be declared or defined repeatedly. Reloading is not supported.
- The identifier of a nested subprogram cannot be the same as the variable name or keyword.
- Declaration before definition is supported. The definition of subprogram declared first must be found in the subsequent declaration block.

Calling Rules of Nested Subprograms

- Nested subprograms can call themselves to achieve recursive calling effects.
- Nested subprograms can call upper-layer subprograms.
- Nested subprograms can call locally declared lower-layer subprograms, but cannot call nested subprograms in lower-layer subprograms.
- Nested subprograms can call subprograms declared earlier than themselves at the same layer.

Example:

```
-- Call itself.
CREATE OR REPLACE PROCEDURE proc_test(var1 int) AS
    PROCEDURE proc_sub(var2 int) IS
    BEGIN
        dbe_output.put_line('var = ' || var2);
        IF var2 > 1 THEN
            proc_sub(var2 - 1);
        END IF;
    END;
BEGIN
    proc_sub(var1);
END;
/
BEGIN
    proc_test(3);
END;
/
-- Output the result.
var = 3
var = 2
var = 1

-- Call the upper-layer subprogram.
CREATE OR REPLACE PROCEDURE proc_test(var1 int) AS
    PROCEDURE procsub_1(var2 int) IS
    BEGIN
    proc_test(var2 - 1);
    END;
BEGIN
    dbe_output.put_line('proc_test var1 = ' || var1);
    IF var1 > 1 THEN
          procsub_1(var1);
    END IF;
END;
/
BEGIN
    proc_test(3);
END;
/
-- Output the result.
proc_test var1 = 3
proc_test var1 = 2
proc_test var1 = 1

-- Call the lower-layer subprogram declared locally.
CREATE OR REPLACE PROCEDURE proc_test() AS
    PROCEDURE proc_sub1 IS
        PROCEDURE proc_sub2 IS
        BEGIN
            dbe_output.put_line('--this is subpragram2 begin');
            dbe_output.put_line('--this is subpragram2 end');
        END;
    BEGIN
    dbe_output.put_line('this is subpragram1 begin');
    proc_sub2();
    dbe_output.put_line('this is subpragram1 end');
    END;
```



```
BEGIN
    dbe_output.put_line('this is a procedure begin');
    proc_sub1();
    dbe_output.put_line('this is a procedure end');
END;
/
BEGIN
    proc_test;
END;
/
-- Output the result.
this is a procedure begin
this is subpragram1 begin
--this is subpragram2 begin
--this is subpragram2 end
this is subpragram1 end
this is a procedure end
```

Variables of Nested Subprograms

Variable types include basic types, record types, TABLE OF types, cursor types, and varray types supported by PL/SQL.

- Accessible variables:
  - Variable declared by itself.
  - Variable declared by the upper-layer subprogram.
- Variable access rules:
  - If a variable does not contain a qualifier, the variable is first searched in the program. If the variable name does not exist, the variable is searched at the upper layer, and so on.
  - If a variable has a qualifier, it is searched in the area of the qualifier. Currently, only one qualifier can be called.

10.6 Basic Statements

During PL/SQL programming, you may define some variables, assign values to variables, and call other stored procedures. This chapter describes basic PL/SQL statements, including variable definition statements, value assignment statements, call statements, and return statements.

NOTE

You are advised not to call the SQL statements containing passwords in the stored procedures because authorized users may view the stored procedure file in the database and password information is leaked. If a stored procedure contains other sensitive information, permission to access this procedure must be configured, preventing information leakage.

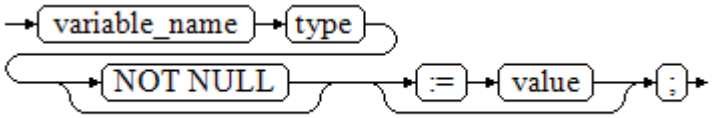
10.6.1 Variable Definition Statements

This section describes the declaration of variables in the PL/SQL and the scope of this variable in codes.

Variable Declaration

For details about the variable declaration syntax, see **Figure 10-3**.

Figure 10-3 declare_variable::=



The above syntax diagram is explained as follows:

- **variable_name** indicates the name of a variable.
- **type** indicates the type of a variable.
- **value** indicates the initial value of the variable. (If the initial value is not given, NULL is taken as the initial value.) **value** can also be an expression.

Examples

```
gaussdb=# DECLARE
    emp_id  INTEGER := 7788; -- Define a variable and assign a value to it.
BEGIN
    emp_id := 5*7784; -- Assign a value to the variable.
END;
/
ANONYMOUS BLOCK EXECUTE
```

In addition to the declaration of basic variable types, **%TYPE** and **%ROWTYPE** can be used to declare variables related to table columns or table structures.

%TYPE Attribute

%TYPE declares a variable to be of the same data type as a previously declared variable (for example, a column in a table). For example, if you want to define the *my_name* variable whose data type is the same as the data type of **firstname** in **employee**, you can define the variable as follows:

```
my_name employee.firstname%TYPE
-- Example
DROP TABLE IF EXISTS employee;
NOTICE:  table "employee" does not exist, skipping
DROP TABLE
CREATE TABLE employee(firstname varchar,secondname varchar);
CREATE TABLE
DECLARE
    my_name employee.firstname%TYPE;
BEGIN
    my_name = 'abc';
    DBE_OUTPUT.PRINT_LINE(my_name);
END;
/
abc
ANONYMOUS BLOCK EXECUTE
```

In this way, you do not need to know the data type of **firstname** in **employee**. In addition, you do not need to change the data type of *my_name* again when the data type of **firstname** changes.

```
TYPE employee_record is record (id INTEGER, firstname VARCHAR2(20));
my_employee employee_record;
my_id my_employee.id%TYPE;
my_id_copy my_id%TYPE;
-- Example
DECLARE
    TYPE employee_record is record (id INTEGER, firstname VARCHAR2(20));
    my_employee employee_record;
```



```
    my_id my_employee.id%TYPE;
    my_id_copy my_id%TYPE;
BEGIN
    my_employee.id := 1;
    my_employee.firstname := 'ab2';
    my_id = 2;
    my_id_copy = 3;
    DBE_OUTPUT.PRINT_LINE(my_employee.id);
    DBE_OUTPUT.PRINT_LINE(my_employee.firstname);
    DBE_OUTPUT.PRINT_LINE(my_id);
    DBE_OUTPUT.PRINT_LINE(my_id_copy);
END;
/
1
ab2
2
3
ANONYMOUS BLOCK EXECUTE
```

%ROWTYPE Attribute

%ROWTYPE declares data types of a set of data. It stores a row of table data or results fetched from a cursor. For example, if you want to define a set of data with the same column names and column data types as the **employee** table, you can define the data as follows:

```
my_employee employee%ROWTYPE
-- Example
DROP TABLE IF EXISTS employee;
DROP TABLE
CREATE TABLE employee(firstname varchar,secondname varchar);
DECLARE
    my_employee employee%ROWTYPE;
BEGIN
    my_employee.firstname := 'ab1';
    my_employee.secondname := 'ab2';
    DBE_OUTPUT.PRINT_LINE(my_employee.firstname);
    DBE_OUTPUT.PRINT_LINE(my_employee.secondname);
END;
/
ab1
ab2
ANONYMOUS BLOCK EXECUTE
```

The attribute can also be used on the cursor. The column names and column data types of this set of data are the same as those of the **employee** table. For the cursor in a package, **%ROWTYPE** can be omitted. **%TYPE** can also reference the type of a column in the cursor. You can define the data as follows:

```
cursor cur is select * from employee;
my_employee cur%ROWTYPE
my_name cur.firstname%TYPE
my_employee2 cur -- For the cursor defined in a package, %ROWTYPE can be omitted.
-- Example
set behavior_compat_options = 'allow_procedure_compile_check';
SET
DROP TABLE IF EXISTS employee;
DROP TABLE
CREATE TABLE employee(firstname varchar,secondname varchar);
CREATE TABLE
CREATE OR REPLACE procedure proc1()
IS
    cursor cur is select * from employee;
    my_employee cur%ROWTYPE;
    my_name cur.firstname%TYPE;
BEGIN
    my_employee.firstname := 'ab1';
    my_employee.secondname := 'ab2';
```



```
    my_name := 'ab3';
    DBE_OUTPUT.PRINT_LINE(my_employee.firstname);
    DBE_OUTPUT.PRINT_LINE(my_employee.secondname);
    DBE_OUTPUT.PRINT_LINE(my_name);
END;
/
CREATE PROCEDURE
```

NOTICE

- **%TYPE** can reference only the type of record variables across packages and cannot reference a column type of the cursor variable.
- If the GUC parameter **behavior_compat_options** is enabled and the value of this parameter contains **allow_procedure_compile_check**, **%TYPE** can reference a column type of the cursor variable.
- **%ROWTYPE** cannot reference the type of a composite variable or a record variable.
- The non-PACKAGE *record%TYPE* cannot be used as the variable type or input/output parameter type.
- *View%ROWTYPE* or **SCHEMA.***view%ROWTYPE* cannot be used as the input/output parameter type.
- The **PACKAGE.***record%TYPE* or **SCHEMA.PACKAGE.***record%TYPE* cannot be used as the input/output parameter type.
- *Table/View*.**column.***column%TYPE* or **SCHEMA.***Table/View*.**column.***column%TYPE* cannot be nested with one or more layers as variable types or input/output parameter type.
- *Record*.**column.***column%TYPE* and **PACKAGE.***record*.**column.***column%TYPE* cannot be nested with a column type of records at one or more layers as the variable type or input/output parameter type.
- The cursor variable of more levels, such as **PACKAGE.***cursor%ROWTYPE* and **SCHEMA.PACKAGE.***cursor%ROWTYPE*, cannot be used as the parameter type or input/output parameter type.

Scope of a Variable

The scope of a variable indicates the accessibility and availability of the variable in code block. In other words, a variable takes effect only within its scope.

- To define a function scope, a variable must declare and create a **BEGIN-END** block in the declaration section. The necessity of such declaration is also determined by block structure, which requires that a variable has different scopes and lifetime during a process.
- A variable can be defined multiple times in different scopes, and inner definition can cover outer one.
- A variable defined in an outer block can also be used in a nested block. However, the outer block cannot access variables in the nested block.

Example:

```
gaussdb=# DECLARE
    emp_id  INTEGER :=7788; -- Define a variable and assign a value to it.
    outer_var  INTEGER :=6688; -- Define a variable and assign a value to it.
BEGIN
```



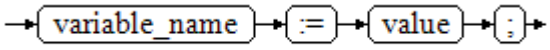
```
    DECLARE
        emp_id INTEGER :=7799; -- Define a variable and assign a value to it.
        inner_var  INTEGER :=6688; -- Define a variable and assign a value to it.
    BEGIN
        dbe_output.print_line('inner emp_id ='||emp_id); -- Display the value 7799.
        dbe_output.print_line('outer_var ='||outer_var); -- Reference a variable of an outer block.
    END;
    dbe_output.print_line('outer emp_id ='||emp_id); -- Display the value 7788.
END;
/
inner emp_id =7799
outer_var =6688
outer emp_id =7788
ANONYMOUS BLOCK EXECUTE
```

10.6.2 Assignment Statements

Variable Syntax

Figure 10-4 shows the syntax diagram for assigning a value to a variable.

Figure 10-4 assignment_value::=



The above syntax diagram is explained as follows:

- **variable_name** indicates the name of a variable.
- **value** can be a value or an expression. The type of *value* must be compatible with the type of *variable_name*.

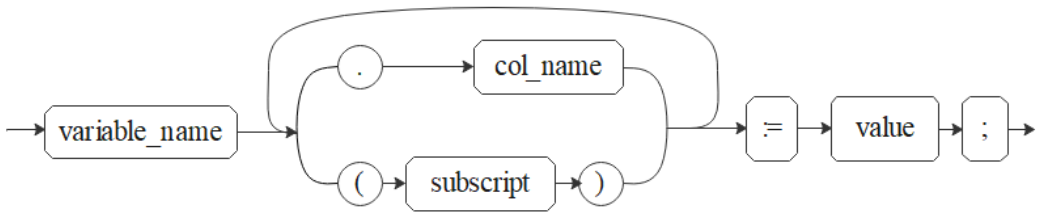
Example:

```
DECLARE
    emp_id  INTEGER := 7788; -- Assignment
BEGIN
    emp_id := 5; -- Assignment
    DBE_OUTPUT.PRINT_LINE(emp_id);
    emp_id := 5*7784;
    DBE_OUTPUT.PRINT_LINE(emp_id);
END;
/
-- The result is as follows:
5
38920
ANONYMOUS BLOCK EXECUTE
```

Nested Value Assignment

Figure 10-5 shows the syntax diagram for assigning a nested value to a variable.

Figure 10-5 nested_assignment_value::=



The syntax in **Figure 10-5** is described as follows:

- **variable_name**: variable name
- **col_name**: column name
- **subscript**: index, which is used for an array variable. The value can be a value or an expression and must be of the INT type.
- **value** can be a value or an expression. The type of *value* must be compatible with the type of *variable_name*.

Example:

```
gaussdb=# CREATE TYPE o1 AS (a int, b int);
CREATE TYPE
gaussdb=# DECLARE
    TYPE r1 is VARRAY(10) of o1;
    emp_id  r1;
BEGIN
    emp_id(1).a := 5;-- Assign a value.
    emp_id(1).b := 5*7784;
END;
/
ANONYMOUS BLOCK EXECUTE
```

NOTICE

In INTO mode, values can be assigned only to the columns at the first layer. Two-dimensional or above arrays are not supported.

INTO/BULK COLLECT INTO

- Values returned by statements in a stored procedure are stored in variables. BULK COLLECT INTO allows some or all returned values to be temporarily stored in an array.
- An empty result set can be returned.

Syntax

```
SELECT select_expressions INTO [STRICT] target FROM ...
SELECT INTO [STRICT] target [FROM ..]
EXECUTE [IMMEDIATE] select_expressions BULK COLLECT INTO target ...
```

The syntax is described as follows:

- **select_expressions**: SQL statement for query. You can use basic SQL commands and INTO clauses to assign the result of a single row or multiple columns to a variable (such as record, row, or scalar variable list).
- **target**: target variable, which can be a record variable, a row variable, or a comma-separated list of simple variables and record/row fields.
- **STRICT** (Optional): If **set behavior_compat_options** is set to **'select_into_return_null'** (disabled by default) and the **STRICT** option is specified, the query must return a non-empty result set. Otherwise, an error is reported: "NO_DATA_FOUND", "TOO_MANY_ROWS" or "QUERY_RETURNED_NO_ROWS". If the **STRICT** option is not specified, the empty result set can be returned.

NOTE

- BULK COLLECT INTO can only assign values to arrays or collections in batches. The collection type uses LIMIT properly to prevent performance deterioration caused by excessive operations on data.
- INTO and BULK COLLECT INTO support only direct nesting of record type value with less than four layers.
- To return an empty result set, you need to enable PostgreSQL compatibility mode during database initialization. Set **set behavior_compat_options** to **'select_into_return_null'**, the compatibility mode is enabled. If the GUC parameter **set behavior_compat_options** is not set, the compatibility mode is disabled.
- For array variables, elements in parentheses () are preferentially identified as index sets. Therefore, expressions with parentheses cannot be written after array variables. For example, **select (1+3) into va(5)** cannot be written as **select into va(5) (1+3)** or **select into va[5] (1+3)**.
- INSERT INTO, UPDATE INTO, DELETE INTO, and EXECUTION INTO cannot return an empty result set.
- Multiple variables are failed to be assigned values because the syntax of the subsequent variables is incorrect.
- BULK COLLECT INTO can be used only in the A-compatible database.
- The IMMEDIATE keyword is used only for syntax compatibility and has no actual meaning.

Example:

```
gaussdb=# DROP TABLE IF EXISTS customers;
NOTICE:  table "customers" does not exist, skipping
DROP TABLE
gaussdb=# CREATE TABLE customers(id int,name varchar);
CREATE TABLE
gaussdb=# INSERT INTO customers VALUES(1,'ab');
gaussdb=# DECLARE
    my_id integer;
BEGIN
    select id into my_id from customers limit 1; -- Assign a value.
END;
/
ANONYMOUS BLOCK EXECUTE
gaussdb=# DECLARE
    type id_list is varray(6) of customers.id%type;
    id_arr id_list;
BEGIN
    select id bulk collect into id_arr from customers order by id DESC limit 20; -- Assign values in batches.
END;
/
ANONYMOUS BLOCK EXECUTE
gaussdb=# CREATE TABLE test(a integer);
CREATE TABLE
gaussdb=# INSERT INTO test VALUES(1);
INSERT 0 1
gaussdb=# CREATE OR REPLACE FUNCTION check_test() RETURNS integer
    language plpgsql
    AS $function$
    DECLARE
        b integer;
    BEGIN
        SELECT INTO b FROM test WHERE a=1; -- Return an empty result set.
            RETURN b;
    END;
    $function$;
CREATE FUNCTION
gaussdb=# SELECT check_test();
 check_test
------------
          1
```



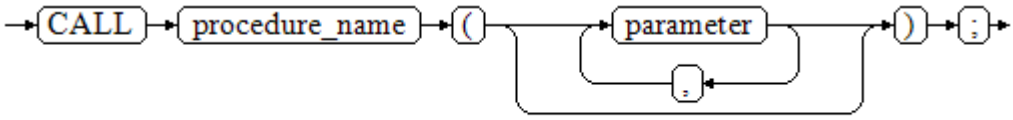
```
(1 row)
gaussdb=#  DECLARE
    TYPE id_list IS varray(6) OF customers.id%type;
    id_arr id_list;
    sql_qry varchar2(150);
BEGIN
    sql_qry := 'SELECT id FROM customers ORDER BY id DESC LIMIT 20';
    EXECUTE IMMEDIATE sql_qry BULK COLLECT INTO id_arr; -- Assign values in batches.
END;
/
ANONYMOUS BLOCK EXECUTE
```

10.6.3 Call Statements

Syntax

Figure 10-6 shows the syntax diagram for calling a clause.

Figure 10-6 call_clause::=



The above syntax diagram is explained as follows:

- **procedure_name** specifies the name of a stored procedure.
- **parameter** specifies the parameters for the stored procedure. You can set no parameter or multiple parameters.

Examples

```
-- Create a table.
gaussdb=# CREATE SCHEMA hr;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA = hr;
SET
gaussdb=# CREATE TABLE staffs
(
  section_id INTEGER,
  salary INTEGER
);
CREATE TABLE
gaussdb=# INSERT INTO staffs VALUES (30, 10);
INSERT 0 1
gaussdb=# INSERT INTO staffs VALUES (30, 20);
INSERT 0 1
-- Create the stored procedure proc_staffs.
gaussdb=# CREATE OR REPLACE PROCEDURE proc_staffs
(
section     NUMBER(6),
salary_sum out NUMBER(8,2),
staffs_count out INTEGER
)
IS
BEGIN
SELECT sum(salary), count(*) INTO salary_sum, staffs_count FROM hr.staffs where section_id = section;
END;
/
CREATE PROCEDURE
```



```
-- Create the stored procedure proc_return.
gaussdb=# CREATE OR REPLACE PROCEDURE proc_return
AS
v_num NUMBER(8,2);
v_sum INTEGER;
BEGIN
proc_staffs(30, v_sum, v_num);  -- Call a statement.
dbe_output.print_line(v_sum||'#'||v_num);
RETURN;  -- Return a statement.
END;
/
CREATE PROCEDURE
-- Call the stored procedure proc_return.
gaussdb=# CALL proc_return();
30#2.00
 proc_return
-------------

(1 row)
-- Drop the stored procedure.
gaussdb=# DROP PROCEDURE proc_staffs;
DROP PROCEDURE
gaussdb=# DROP PROCEDURE proc_return;
DROP PROCEDURE
```

10.7 Dynamic Statements

10.7.1 Executing Dynamic Query Statements

This section describes how to execute dynamic query statements. GaussDB provides two methods for dynamic query: EXECUTE IMMEDIATE and OPEN FOR. The former dynamically executes SELECT statements while the latter combines the use of cursors. If you need to store query results in a dataset, use OPEN FOR.

EXECUTE IMMEDIATE

Figure 10-7 shows the syntax diagram.

Figure 10-7 EXECUTE IMMEDIATE dynamic_select_clause::=

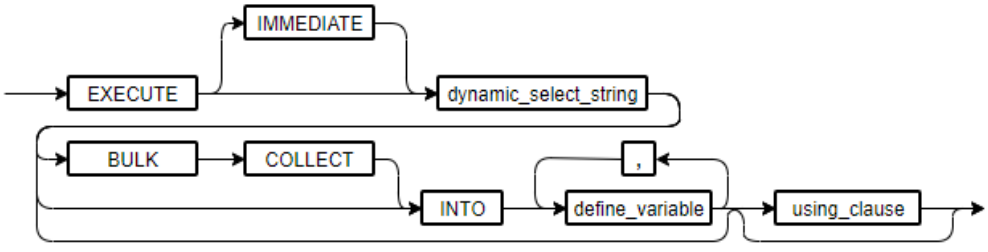
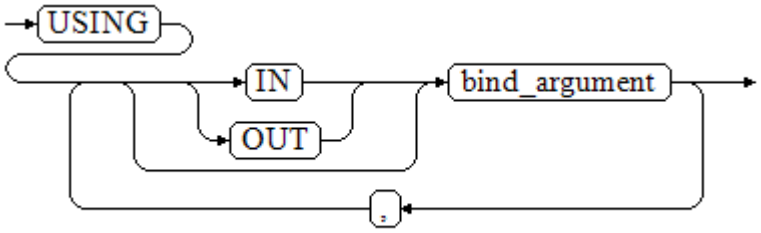


Figure 10-8 shows the syntax diagram for **using_clause**.

Figure 10-8 using_clause::=



The above syntax diagram is explained as follows:

- **define_variable**: specifies a variable to store query results.
- **USING IN** *bind_argument*: specifies the variable whose value is passed to the dynamic SQL statement. The variable is used when a dynamic placeholder exists in *dynamic_select_string*.
- **USING OUT** *bind_argument*: specifies the variable that stores a value returned by the dynamic SQL statement.

NOTICE

- In query statements, INTO and OUT cannot coexist.
- A placeholder name starts with a colon (:) followed by digits, characters, or strings, corresponding to *bind_argument* in the **USING** clause.
- *bind_argument* can only be a value, variable, or expression. It cannot be a database object such as a table name, column name, and data type. That is, *bind_argument* cannot be used to transfer schema objects for dynamic SQL statements. If a stored procedure needs to transfer database objects through *bind_argument* to construct dynamic SQL statements (generally, DDL statements), you are advised to use double vertical bars (||) to concatenate *dynamic_select_clause* with a database object.
- A dynamic PL/SQL block allows duplicate placeholders. That is, a placeholder can correspond to only one *bind_argument* in the USING clause. When the GUC parameter **behavior_compat_options** is set to **dynamic_sql_compat**, the bind arguments in the USING clause are matched in sequence based on the placeholder sequence. Duplicate placeholders will not be identified as the same placeholder.
- The IMMEDIATE keyword is used only for syntax compatibility and has no actual meaning.

Example:

```
gaussdb=# DROP SCHEMA IF EXISTS hr CASCADE;
NOTICE:  drop cascades to table staffs
DROP SCHEMA
gaussdb=# CREATE SCHEMA hr;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA = hr;
SET
gaussdb=# CREATE TABLE staffs
(
  staff_id NUMBER,
  first_name VARCHAR2,
```



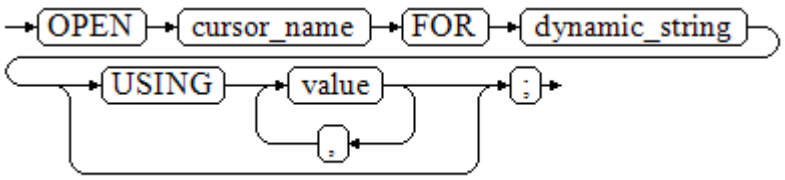
```
 salary NUMBER
);
CREATE TABLE
gaussdb=# INSERT INTO staffs VALUES (200, 'mike', 5800);
INSERT 0 1
gaussdb=# INSERT INTO staffs VALUES (201, 'lily', 3000);
INSERT 0 1
gaussdb=# INSERT INTO staffs VALUES (202, 'john', 4400);
INSERT 0 1
-- Retrieve values from dynamic statements (INTO clause).
gaussdb=# DECLARE
   staff_count  VARCHAR2(20);
BEGIN
   EXECUTE IMMEDIATE 'select count(*) from hr.staffs'
      INTO staff_count;
   dbe_output.print_line(staff_count);
END;
/
3
ANONYMOUS BLOCK EXECUTE
-- Pass and retrieve values (the INTO clause is used before the USING clause).
gaussdb=# CREATE OR REPLACE PROCEDURE dynamic_proc
AS
   staff_id     NUMBER(6) := 200;
   first_name   VARCHAR2(20);
   salary       NUMBER(8,2);
BEGIN
   EXECUTE IMMEDIATE 'select first_name, salary from hr.staffs where staff_id = :1'
      INTO first_name, salary
      USING IN staff_id;
   dbe_output.print_line(first_name || ' ' || salary);
END;
/
CREATE PROCEDURE
-- Call the stored procedure.
gaussdb=# CALL dynamic_proc();
mike 5800.00
 dynamic_proc
--------------

(1 row)
-- Delete the stored procedure.
gaussdb=# DROP PROCEDURE dynamic_proc;
DROP PROCEDURE
```

OPEN FOR

Dynamic query statements can be executed by using OPEN FOR to open dynamic cursors.

Figure 10-9 shows the syntax diagram.

Figure 10-9 open_for::=



Parameter description:

- **cursor_name**: specifies the name of the cursor to be opened.
- **dynamic_string**: specifies the dynamic query statement.
- **USING** *value*: applies when a placeholder exists in dynamic_string.

For the use of cursors, see **Cursors**.

Example:

```
gaussdb=# CREATE SCHEMA hr;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA = hr;
SET
gaussdb=# CREATE TABLE staffs
(
  section_id NUMBER,
  first_name VARCHAR2,
  phone_number VARCHAR2,
  salary NUMBER
  );
CREATE TABLE
gaussdb=# INSERT INTO staffs VALUES (30, 'mike', '13567829252', 5800);
INSERT 0 1
gaussdb=# INSERT INTO staffs VALUES (40, 'john', '17896354637', 4000);
INSERT 0 1
gaussdb=# DECLARE
    name          VARCHAR2(20);
    phone_number  VARCHAR2(20);
    salary        NUMBER(8,2);
    sqlstr        VARCHAR2(1024);

    TYPE app_ref_cur_type IS REF CURSOR; -- Define the cursor type.
    my_cur app_ref_cur_type; -- Define the cursor variable.

BEGIN
    sqlstr := 'select first_name,phone_number,salary from hr.staffs
        where section_id = :1';
    OPEN my_cur FOR sqlstr USING '30'; -- Open the cursor. USING is optional.
    FETCH my_cur INTO name, phone_number, salary; -- Retrieve the data.
    WHILE my_cur%FOUND LOOP
          dbe_output.print_line(name||'#'||phone_number||'#'||salary);
          FETCH my_cur INTO name, phone_number, salary;
    END LOOP;
    CLOSE my_cur; -- Close the cursor.
END;
/
mike#13567829252#5800.00
ANONYMOUS BLOCK EXECUTE
```

10.7.2 Executing Dynamic Non-Query Statements

Syntax

Figure 10-10 shows the syntax diagram.

Figure 10-10 noselect::=

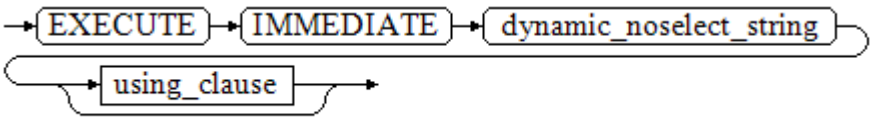
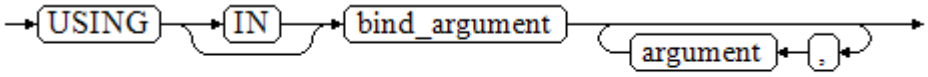


Figure 10-11 shows the syntax diagram for **using_clause**.

Figure 10-11 using_clause::=



The above syntax diagram is explained as follows:

USING IN *bind_argument* is used to specify the variable whose value is passed to the dynamic SQL statement. The variable is used when a placeholder exists in *dynamic_noselect_string*. That is, a placeholder is replaced by the corresponding *bind_argument* when a dynamic SQL statement is executed. Note that *bind_argument* can only be a value, variable, or expression, and cannot be a database object such as a table name, column name, and data type. If a stored procedure needs to transfer database objects through *bind_argument* to construct dynamic SQL statements (generally, DDL statements), you are advised to use double vertical bars (||) to concatenate *dynamic_select_clause* with a database object. In addition, a dynamic PL/SQL block allows duplicate placeholders. That is, a placeholder can correspond to only one *bind_argument*. When the GUC parameter **behavior_compat_options** is set to **dynamic_sql_compat**, the bind arguments in the USING clause are matched in sequence based on the placeholder sequence. Duplicate placeholders will not be identified as the same placeholder.

Examples

```
-- Create a table.
gaussdb=# CREATE TABLE sections_t1
(
  section       NUMBER(4) ,
  section_name  VARCHAR2(30),
  manager_id    NUMBER(6),
  place_id      NUMBER(4)
);
CREATE TABLE
-- Declare a variable.
gaussdb=# DECLARE
  section       NUMBER(4) := 280;
  section_name  VARCHAR2(30) := 'Info support';
  manager_id    NUMBER(6) := 103;
  place_id      NUMBER(4) := 1400;
  new_colname   VARCHAR2(10) := 'sec_name';
BEGIN
-- Execute the query.
   EXECUTE IMMEDIATE 'insert into sections_t1 values(:1, :2, :3, :4)'
      USING section, section_name, manager_id,place_id;
-- Execute the query (duplicate placeholders).
   EXECUTE IMMEDIATE 'insert into sections_t1 values(:1, :2, :3, :1)'
      USING section, section_name, manager_id;
-- Run the ALTER statement. (You are advised to use double vertical bars (||) to concatenate the dynamic
DDL statement with a database object.)
   EXECUTE IMMEDIATE 'alter table sections_t1 rename section_name to ' || new_colname;
END;
/
ANONYMOUS BLOCK EXECUTE
-- Query data.
gaussdb=# SELECT * FROM sections_t1;
 section |   sec_name   | manager_id | place_id
---------+--------------+------------+----------
     280 | Info support |        103 |     1400
     280 | Info support |        103 |      280
```



```
(2 rows)
-- Drop the table.
gaussdb=# DROP TABLE sections_t1;
DROP TABLE
```

10.7.3 Dynamically Calling Stored Procedures

This section describes how to dynamically call store procedures. You must use anonymous statement blocks to package stored procedures or statement blocks and append IN and OUT behind the EXECUTE IMMEDIATE...USING statement to input and output parameters.

Syntax

Figure 10-12 shows the syntax diagram.

Figure 10-12 call_procedure::=

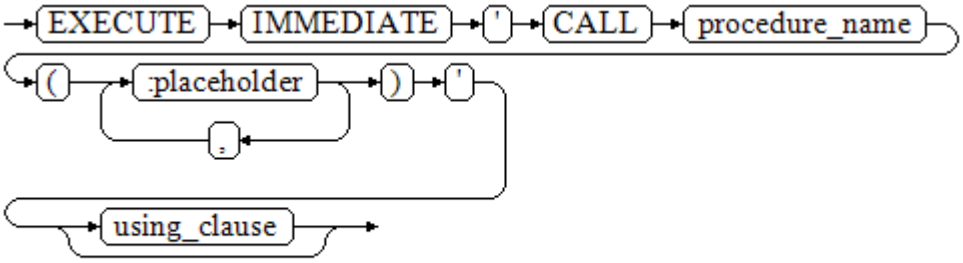
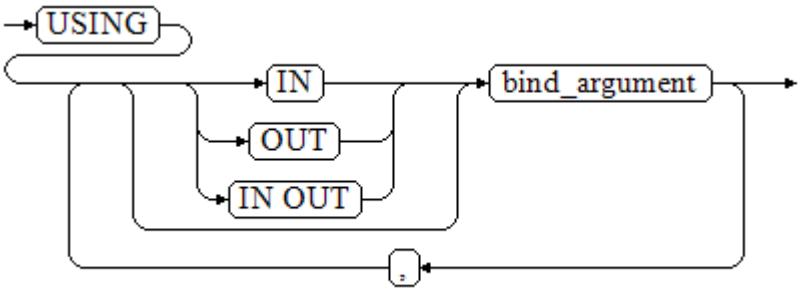


Figure 10-13 shows the syntax diagram for **using_clause**.

Figure 10-13 using_clause::=



The above syntax diagram is explained as follows:

- **CALL procedure_name**: calls the stored procedure.
- **[:placeholder1,:placeholder2,...]**: specifies the placeholder list of the stored procedure parameters. The numbers of the placeholders and parameters are the same.
- **USING [IN|OUT|IN OUT]bind_argument**: specifies where the variable passed to the stored procedure parameter value is stored. The modifiers in front of **bind_argument** and of the corresponding parameter are the same.

- Overloaded functions or stored procedures with placeholders cannot be called.
- When a stored procedure is called, => cannot be used to skip parameters.

Example:

```
-- Create the stored procedure proc_add.
gaussdb=# CREATE OR REPLACE PROCEDURE proc_add
(
    param1    in   INTEGER,
    param2    out  INTEGER,
    param3    in   INTEGER
)
AS
BEGIN
   param2:= param1 + param3;
END;
/
CREATE PROCEDURE
gaussdb=# DECLARE
    input1 INTEGER:=1;
    input2 INTEGER:=2;
    statement  VARCHAR2(200);
    param2     INTEGER;
BEGIN
   -- Declare the call statement.
    statement := 'call proc_add(:col_1, :col_2, :col_3)';
   -- Execute the statement.
    EXECUTE IMMEDIATE statement
        USING IN input1, OUT param2, IN input2;
    dbe_output.print_line('result is: '||to_char(param2));
END;
/
result is: 3
ANONYMOUS BLOCK EXECUTE
-- Delete the stored procedure.
gaussdb=# DROP PROCEDURE proc_add;
DROP PROCEDURE
```

10.7.4 Dynamically Calling Anonymous Blocks

This section describes how to execute anonymous blocks in dynamic statements. Append IN and OUT behind the EXECUTE IMMEDIATE...USING statement to input and output parameters.

Syntax

Figure 10-14 shows the syntax diagram.

Figure 10-14 call_anonymous_block::=

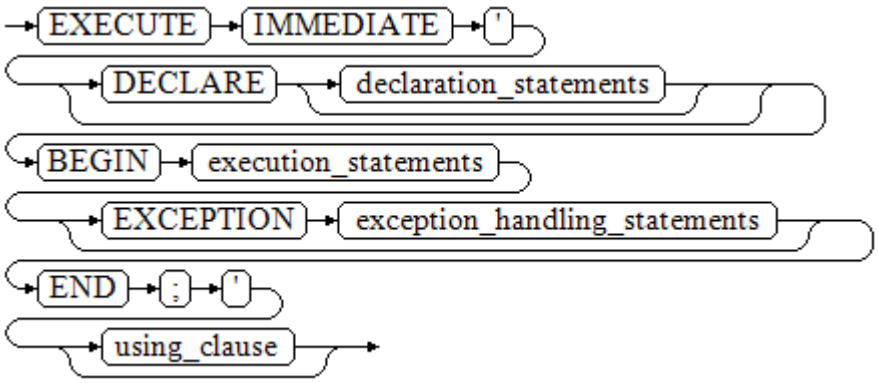
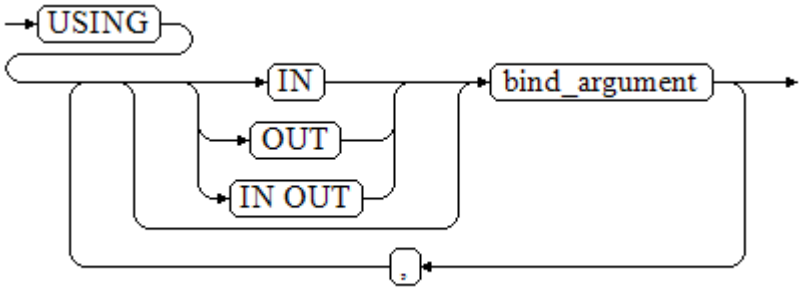


Figure 10-15 shows the syntax diagram for **using_clause**.

Figure 10-15 using_clause::=



The above syntax diagram is explained as follows:

- The execute part of an anonymous block starts with a BEGIN statement, has a break with an END statement, and ends with a semicolon (;).
- **USING [IN|OUT|IN OUT]bind_argument**: specifies where the variable passed to the stored procedure parameter value is stored. The modifiers in front of **bind_argument** and of the corresponding parameter are the same.
- The input and output parameters in the middle of an anonymous block are designated by placeholders. The numbers of the placeholders and parameters are the same. The sequences of the parameters corresponding to the placeholders and the USING parameters are the same.

  NOTE

  - Parameters can be bound only when SQL statements and stored procedures are called in anonymous blocks. For example, expressions and cursors are used in anonymous blocks, and dynamic statements are called in nested mode in anonymous blocks.
  - Output parameters cannot be bound when the SELECT INTO statement in an anonymous block calls FUNCTION/PROCEDURE that contains output parameters.
  - Variables declared in an anonymous block and binding parameters cannot be used in the same statement at the same time.
  - The PERFORM keyword cannot be used to call a stored procedure when parameters are bound.
  - When a stored procedure is called, only bound parameters can be used as input and output parameters. Expressions (for example, 1+:va) cannot be used as input and output parameters.
  - When the bound input parameter type is refcursor, the modification in the stored procedure is isolated from the input parameter.

Example:

```
gaussdb=# DROP SCHEMA IF EXISTS hr CASCADE;
gaussdb=# CREATE SCHEMA hr;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA = hr;
SET
gaussdb=# CREATE TABLE staffs
(
  staff_id NUMBER,
  first_name VARCHAR2,
  salary NUMBER
);
```



```
CREATE TABLE
gaussdb=# INSERT INTO staffs VALUES (200, 'mike', 5800);
INSERT 0 1
gaussdb=# INSERT INTO staffs VALUES (201, 'lily', 3000);
INSERT 0 1
gaussdb=# INSERT INTO staffs VALUES (202, 'john', 4400);
INSERT 0 1

-- Create overloaded functions.
gaussdb=# CREATE OR REPLACE PACKAGE pkg1
IS
  PROCEDURE plus(var1 in int, var2 int, var3 out int);
  PROCEDURE plus(var1 in out int);
END pkg1;
/
CREATE PACKAGE
gaussdb=# CREATE OR REPLACE PACKAGE BODY pkg1 IS
  PROCEDURE plus(var1 in int, var2 int, var3 out int)
  AS
  BEGIN
    var3 = var1 + var2 + 1;
  END;
  PROCEDURE plus(var1 in out int)
  AS
  BEGIN
    var1 = var1 + 1;
  END;
END pkg1;
/
CREATE PACKAGE BODY

-- Create the stored procedure dynamic_proc.
gaussdb=# CREATE OR REPLACE PROCEDURE dynamic_proc
AS
  staff_id     NUMBER(6) := 200;
  first_name   VARCHAR2(20);
  salary       NUMBER(8,2);
BEGIN
   -- Execute the anonymous block.
   EXECUTE IMMEDIATE 'begin select first_name, salary into :first_name, :salary from hr.staffs where
staff_id= :dno; end;'
      USING OUT first_name, OUT salary, IN staff_id;
  dbe_output.print_line(first_name|| ' ' || salary);
END;
/
CREATE PROCEDURE
-- Create a stored procedure to call overloaded functions.
gaussdb=# CREATE OR REPLACE PROCEDURE dynamic_proc1
AS
    v_sql  VARCHAR2(200);
    var1   NUMBER(6) := 1;
    var2   NUMBER(6) := 2;
    var3   NUMBER(6);
BEGIN
    v_sql := 'begin pkg1.plus(:1, :2, :3); end;';
    EXECUTE IMMEDIATE v_sql USING var1, var2, out var3;
    dbe_output.print_line('var3: ' || var3);
END;
/
CREATE PROCEDURE

-- Call the stored procedure.
gaussdb=# CALL dynamic_proc();
mike 5800.00
 dynamic_proc
--------------

(1 row)
gaussdb=# CALL dynamic_proc1();
```



```
var3: 4
 dynamic_proc1
---------------

(1 row)
-- Delete the stored procedure.
gaussdb=# DROP PROCEDURE dynamic_proc;
DROP PROCEDURE
gaussdb=# DROP PROCEDURE dynamic_proc1;
DROP PROCEDURE
```

10.8 Control Statements

10.8.1 RETURN Statements

In GaussDB, data can be returned in either of the following ways: **RETURN**, **RETURN NEXT**, or **RETURN QUERY**. **RETURN NEXT** and **RETURN QUERY** are used only for functions and cannot be used for stored procedures.

10.8.1.1 RETURN

Syntax

Figure 10-16 shows the syntax diagram for a return statement.

Figure 10-16 return_clause::=

The above syntax diagram is explained as follows:

Assign the result returned by a stored procedure or function to the caller.

Examples

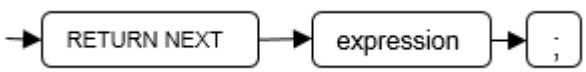
See **Examples** for call statement examples.

10.8.1.2 RETURN NEXT and RETURN QUERY

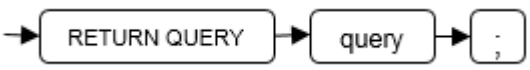
Syntax

When creating a function, specify **SETOF** *datatype* for the return values.

return_next_clause::=



return_query_clause::=



The above syntax diagram is explained as follows:

If a function needs to return a result set, use RETURN NEXT or RETURN QUERY to add results to the result set, and then continue to execute the next statement of the function. As the RETURN NEXT or RETURN QUERY statement is executed repeatedly, more and more results will be added to the result set. After the function is executed, all results are returned.

RETURN NEXT can be used for scalar and compound data types.

RETURN QUERY has a variant RETURN QUERY EXECUTE. You can add dynamic queries and add parameters to the queries by USING.

Examples

```
gaussdb=# DROP TABLE t1;
gaussdb=# CREATE TABLE t1(a int);
gaussdb=# INSERT INTO t1 VALUES(1),(10);

--RETURN NEXT
gaussdb=# CREATE OR REPLACE FUNCTION fun_for_return_next() RETURNS SETOF t1 AS $$
DECLARE
  r t1%ROWTYPE;
BEGIN
  FOR r IN select * from t1
  LOOP
    RETURN NEXT r;
  END LOOP;
  RETURN;
END;
$$ LANGUAGE plpgsql;
gaussdb=# call fun_for_return_next();
 a
---
 1
 10
(2 rows)

-- RETURN QUERY
gaussdb=# CREATE OR REPLACE FUNCTION fun_for_return_query() RETURNS SETOF t1 AS $$
DECLARE
  r t1%ROWTYPE;
BEGIN
  RETURN QUERY select * from t1;
END;
$$
language plpgsql;
gaussdb=# call fun_for_return_query();
 a
---
 1
 10
(2 rows)
```

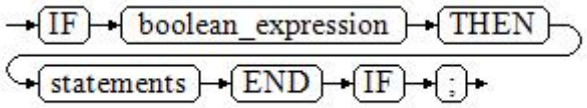
10.8.2 Conditional Statements

Conditional statements are used to decide whether given conditions are met. Operations are executed based on the decisions made.

GaussDB supports five usages of IF:

- IF_THEN

Figure 10-17 IF_THEN::=



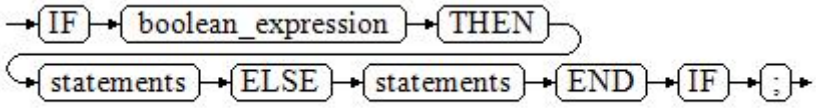
IF_THEN is the simplest form of IF. If the condition is true, statements are executed. If it is false, they are skipped.

Example:

```
gaussdb=# DECLARE
    v_user_id integer default 1;
BEGIN
IF v_user_id <> 0 THEN
    raise info 'v_user_id is NOT 0';

END IF;
END;
/
INFO:  v_user_id is NOT 0
```

- IF_THEN_ELSE

Figure 10-18 IF_THEN_ELSE::=



IF_THEN_ELSE has an ELSE branch and can be executed if the condition is false.

Example:

```
gaussdb=# DECLARE
    v_user_id integer default 1;
BEGIN
    IF v_user_id <> 0 THEN
        raise info 'v_user_id is NOT 0';
    ELSE
        raise info 'v_user_id is 0';
    END IF;
END;
/
INFO:  v_user_id is NOT 0
```

- IF_THEN_ELSE IF

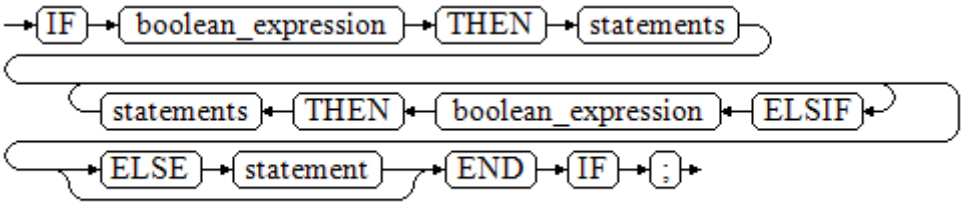
  IF statements can be nested in the following way:

```
gaussdb=# DECLARE
    v_user_id integer default 1;
BEGIN
    IF v_user_id = 0 THEN
        raise info 'v_user_id is  0';
    ELSE
        IF v_user_id > 0 THEN
            raise info 'v_user_id > 0';
        END IF;
    END IF;
END;
/
INFO:  v_user_id > 0
```


Actually, this is a way of an IF statement nesting in the ELSE part of another IF statement. Therefore, an END IF statement is required for each nesting IF statement and another END IF statement is required to end the parent IF-ELSE statement. To set multiple options, use the following form:

- IF_THEN_ELSIF_ELSE

  Figure 10-19 IF_THEN_ELSIF_ELSE::=

  Example:

```
gaussdb=# DECLARE
    v_user_id integer default NULL;
BEGIN
    IF v_user_id = 0 THEN
        raise info 'v_user_id is  0';
    ELSIF v_user_id > 0 THEN
        raise info 'v_user_id > 0';
    ELSIF v_user_id < 0 THEN
        raise info 'v_user_id < 0';
    ELSE
        raise info 'v_user_id is NULL';
    END IF;
END;
/
INFO:  v_user_id is NULL
```

- IF_THEN_ELSEIF_ELSE

  ELSEIF is an alias of ELSIF.

  Example:

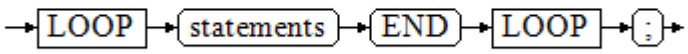
```
gaussdb=# CREATE OR REPLACE PROCEDURE proc_control_structure(i in integer)
AS
    BEGIN
        IF i > 0 THEN
            raise info 'i:% is greater than 0. ',i;
        ELSIF i < 0 THEN
            raise info 'i:% is smaller than 0. ',i;
        ELSE
            raise info 'i:% is equal to 0. ',i;
        END IF;
        RETURN;
    END;
/
CREATE PROCEDURE

gaussdb=# CALL proc_control_structure(3);
INFO:  i:3 is greater than 0.
 proc_control_structure
------------------------


(1 row)

-- Delete the stored procedure.
gaussdb=# DROP PROCEDURE proc_control_structure;
DROP PROCEDURE
```


10.8.3 Loop Statements

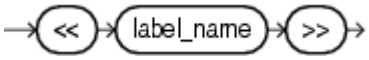
Simple LOOP Statements

Syntax diagram

Figure 10-20 loop::=



label declaration ::=



Example

```
gaussdb=# CREATE OR REPLACE PROCEDURE proc_loop(i in integer, count out integer)
AS
    BEGIN
        count:=0;
        LOOP
        IF count > i THEN
            raise info 'count is %. ', count;
            EXIT;
        ELSE
            count:=count+1;
        END IF;
        END LOOP;
    END;
/
CREATE PROCEDURE

gaussdb=# CALL proc_loop(10,5);
INFO:  count is 11.
 count
-------
    11
(1 row)
```

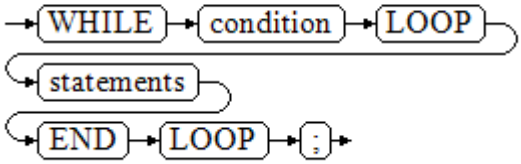
NOTICE

The loop must be exploited together with EXIT; otherwise, a dead loop occurs.

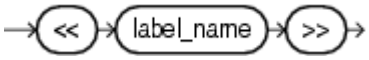
WHILE_LOOP Statements

Syntax diagram

Figure 10-21 while_loop::=



label declaration ::=



If the conditional expression is true, a series of statements in the WHILE statement are repeatedly executed and the condition is decided each time the loop body is executed.

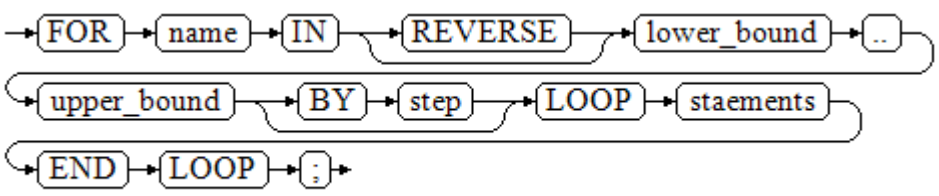
Example

```
gaussdb=# CREATE TABLE integertable(c1 integer) ;
CREATE TABLE
gaussdb=# CREATE OR REPLACE PROCEDURE proc_while_loop(maxval in integer)
AS
    DECLARE
    i int :=1;
    BEGIN
        WHILE i < maxval LOOP
            INSERT INTO integertable VALUES(i);
            i:=i+1;
        END LOOP;
    END;
/
CREATE PROCEDURE

-- Call the stored procedure.
gaussdb=# CALL proc_while_loop(10);
 proc_while_loop
-----------------

(1 row)

-- Delete the stored procedure and table.
gaussdb=# DROP PROCEDURE proc_while_loop;
DROP PROCEDURE
gaussdb=# DROP TABLE integertable;
DROP TABLE
```

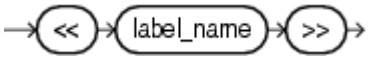
FOR_LOOP (*Integer variable*) *Statement*

Syntax diagram

Figure 10-22 for_loop::=



label declaration ::=



NOTE

- The variable *name* is automatically defined as the integer type and exists only in this loop. The variable name falls between lower_bound and upper_bound.
- When the keyword REVERSE is used, the lower bound must be greater than or equal to the upper bound; otherwise, the loop body is not executed.

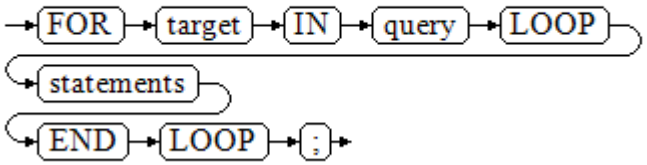
Example

```
-- Loop from 0 to 5.
gaussdb=# CREATE OR REPLACE PROCEDURE proc_for_loop()
AS
    BEGIN
    FOR I IN 0..5 LOOP
        DBE_OUTPUT.PRINT_LINE('It is '||to_char(I) || ' time;') ;
    END LOOP;
END;
/
CREATE PROCEDURE

-- Call the stored procedure.
gaussdb=# CALL proc_for_loop();
It is 0 time;
It is 1 time;
It is 2 time;
It is 3 time;
It is 4 time;
It is 5 time;
 proc_for_loop
---------------

(1 row)

-- Delete the stored procedure.
gaussdb=# DROP PROCEDURE proc_for_loop;
DROP PROCEDURE
```

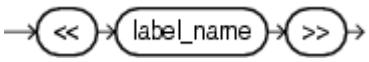
FOR_LOOP Query Statements

Syntax diagram

Figure 10-23 for_loop_query::=



label declaration ::=



NOTE

The variable *target* is automatically defined, its type is the same as that in the query result, and it is valid only in this loop. The target value is the query result.

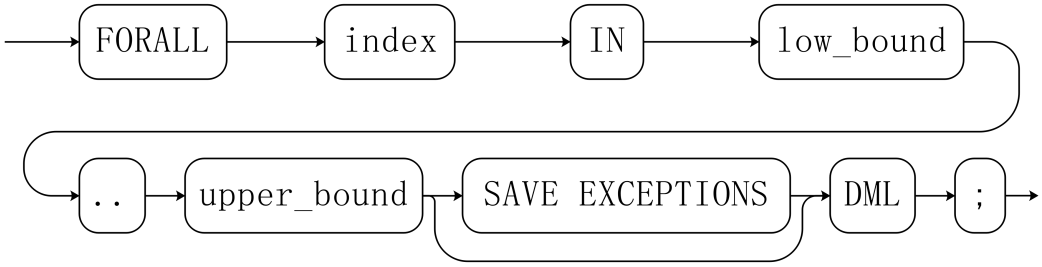
Example

```
-- Display the query result from the loop.
gaussdb=# CREATE OR REPLACE PROCEDURE proc_for_loop_query()
AS
    record VARCHAR2(50);
BEGIN
    FOR record IN SELECT spcname FROM pg_tablespace LOOP
    dbe_output.print_line(record);
    END LOOP;
END;
/
CREATE PROCEDURE

-- Call the stored procedure.
gaussdb=# CALL proc_for_loop_query();
pg_default
pg_global
 proc_for_loop_query
---------------------

(1 row)

-- Delete the stored procedure.
gaussdb=# DROP PROCEDURE proc_for_loop_query;
DROP PROCEDURE
```

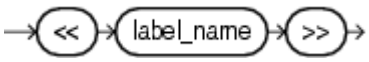
FORALL Batch Query Statements

Syntax diagram

Figure 10-24 forall::=



label declaration ::=



NOTE

- The variable *index* is automatically defined as the integer type and exists only in this loop. The index value falls between low_bound and upper_bound.
- If **SAVE EXCEPTIONS** is specified, exceptions occurred during DML execution in the loop body are saved in **SQL&BULK_EXCEPTIONS** and an exception is thrown after the execution is complete. If there is no abnormal execution result in the loop, the loop will not be rolled back in the current subtransaction.

Example


```
gaussdb=# CREATE TABLE hdfs_t1 (
  title NUMBER(6),
  did VARCHAR2(20),
  data_period VARCHAR2(25),
  kind VARCHAR2(25),
  interval VARCHAR2(20),
  time DATE,
  isModified VARCHAR2(10)
);
CREATE TABLE

gaussdb=# INSERT INTO hdfs_t1 VALUES( 8, 'Donald', 'OConnell', 'DOCONNEL', '650.507.9833',
to_date('21-06-1999', 'dd-mm-yyyy'), 'SH_CLERK' );
INSERT 0 1

gaussdb=# CREATE OR REPLACE PROCEDURE proc_forall()
AS
BEGIN
    FORALL i IN 100..120
        update hdfs_t1 set title = title + 100*i;
END;
/
CREATE PROCEDURE

-- Call the stored procedure.
gaussdb=# CALL proc_forall();
 proc_forall
-------------

(1 row)

-- Query the invocation result of the stored procedure.
gaussdb=# SELECT * FROM hdfs_t1;
 title  |  did   | data_period |   kind   |   interval   |        time         | ismodified
--------+--------+-------------+----------+--------------+---------------------+------------
 231008 | Donald | OConnell    | DOCONNEL | 650.507.9833 | 1999-06-21 00:00:00 | SH_CLERK
(1 row)

-- Delete the stored procedure and table.
gaussdb=# DROP PROCEDURE proc_forall;
DROP PROCEDURE

gaussdb=# DROP TABLE hdfs_t1;
DROP TABLE
```

10.8.4 Branch Statements

Syntax

Figure 10-25 shows the syntax diagram for a branch statement.

Figure 10-25 case_when::=

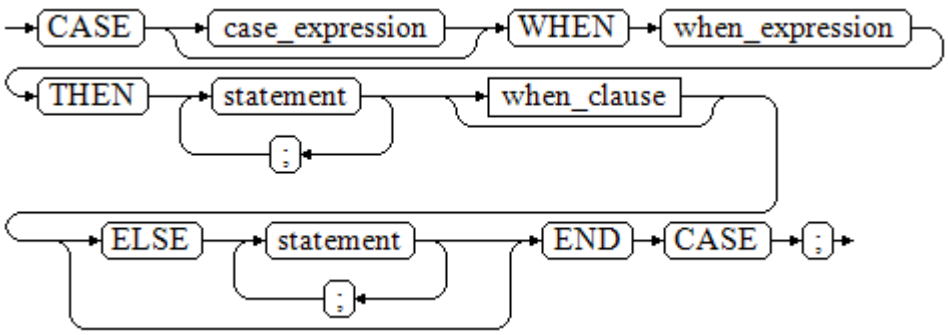
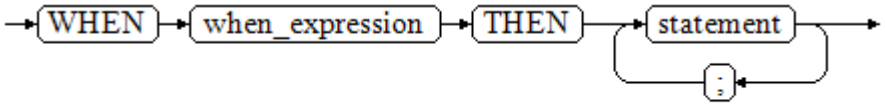


Figure 10-26 shows the syntax diagram for when_clause.

Figure 10-26 when_clause::=



Parameter description:

- *case_expression*: specifies the variable or expression.
- *when_expression*: specifies the constant or conditional expression.
- *statement*: specifies the statement to be executed.

Examples

```
gaussdb=# CREATE OR REPLACE PROCEDURE proc_case_branch(pi_result in integer, pi_return out integer)
AS
   BEGIN
      CASE pi_result
         WHEN 1 THEN
            pi_return := 111;
         WHEN 2 THEN
            pi_return := 222;
         WHEN 3 THEN
            pi_return := 333;
         WHEN 6 THEN
            pi_return := 444;
         WHEN 7 THEN
            pi_return := 555;
         WHEN 8 THEN
            pi_return := 666;
         WHEN 9 THEN
            pi_return := 777;
         WHEN 10 THEN
            pi_return := 888;
         ELSE
            pi_return := 999;
      END CASE;
      raise info 'pi_return : %',pi_return ;
END;
/
CREATE PROCEDURE

gaussdb=# CALL proc_case_branch(3,0);
INFO:  pi_return : 333
 pi_return
-----------
       333
(1 row)

-- Delete the stored procedure.
gaussdb=# DROP PROCEDURE proc_case_branch;
DROP PROCEDURE
```

10.8.5 NULL Statements

In PL/SQL programs, NULL statements are used to indicate "nothing should be done", equal to placeholders. They grant meanings to some statements and improve program readability.

Syntax

The following shows example use of NULL statements:

```
DECLARE
   ...
BEGIN
   ...
   IF v_num IS NULL THEN
      NULL; -- No data needs to be processed.
   END IF;
END;
/
```

Parameter description:

- *v_num*: specifies the variable or expression.

Example

```
gaussdb=# DECLARE
   v_num integer default NULL;
BEGIN
   IF v_num IS NOT NULL THEN
      raise info 'v_num is NULL';
   ELSE
      NULL; -- No data needs to be processed.
   END IF;
END;
/
ANONYMOUS BLOCK EXECUTE
```

10.8.6 Error Trapping Statements

By default, any error occurring in a PL/SQL function aborts execution of the function, and indeed of the surrounding transaction as well. You can trap errors and restore from them by using a **BEGIN** block with an **EXCEPTION** clause. The syntax is an extension of the normal syntax for a **BEGIN** block:

```
[<<label>>]
[DECLARE
   declarations]
BEGIN
   statements
EXCEPTION
   WHEN condition [OR condition ...] THEN
      handler_statements
   [WHEN condition [OR condition ...] THEN
      handler_statements
   ...]
END;
```

If no error occurs, this form of enclosing block simply executes all the statements, and then control passes to the next statement after **END**. But if an error occurs within the statements, further processing of the statements is abandoned, and control passes to the **EXCEPTION** list. The list is searched for the first condition matching the error that occurred. If a match is found, the corresponding **handler_statements** are executed, and then control passes to the next statement after **END**. If no match is found, the error propagates out as though the **EXCEPTION** clause were not there at all: Error codes can be used to catch other error codes of the same type.

The error can be caught by an enclosing block with **EXCEPTION**, or if there is none it aborts processing of the function.

The condition names can be any of those shown in SQL standard error codes. The special condition name **OTHERS** matches every error type except **QUERY_CANCELED**.

If a new error occurs within the selected **handler_statements**, it cannot be caught by this **EXCEPTION** clause, but is propagated out. A surrounding **EXCEPTION** clause could catch it.

When an error is caught by an **EXCEPTION** clause, the local variables of the PL/SQL function remain as they were when the error occurred, but all changes to persistent database state within the block are rolled back.

Example:

```
gaussdb=# CREATE TABLE mytab(id INT,firstname VARCHAR(20),lastname VARCHAR(20)) ;
CREATE TABLE

gaussdb=# INSERT INTO mytab(firstname, lastname) VALUES('Tom', 'Jones');
INSERT 0 1

gaussdb=# CREATE FUNCTION fun_exp() RETURNS INT
AS $$
DECLARE
    x INT :=0;
    y INT;
BEGIN
    UPDATE mytab SET firstname = 'Joe' WHERE lastname = 'Jones';
    x := x + 1;
    y := x / 0;
EXCEPTION
    WHEN division_by_zero THEN
        RAISE NOTICE 'caught division_by_zero';
        RETURN x;
END;$$
LANGUAGE plpgsql;
CREATE FUNCTION

gaussdb=# call fun_exp();
NOTICE:  caught division_by_zero
 fun_exp
---------
       1
(1 row)

gaussdb=# select * from mytab;
 id | firstname | lastname
----+-----------+----------
    | Tom       | Jones
(1 row)

gaussdb=# DROP FUNCTION fun_exp();
DROP FUNCTION

gaussdb=# DROP TABLE mytab;
DROP TABLE
```

When control reaches the assignment to **y**, it will fail with a **division_by_zero** error. This will be caught by the **EXCEPTION** clause. The value returned in the **RETURN** statement will be the incremented value of **x**.

NOTE

- A block containing an **EXCEPTION** clause is more expensive to enter and exit than a block without one. Therefore, do not use **EXCEPTION** without need.
- In the following scenarios, the processing exception cannot be captured and the entire stored procedure is rolled back: 1. The node is faulty. 2. The thread of the node involved in the stored procedure exits due to a network fault. In addition, the structure of the source data is inconsistent with that of the target table in the COPY FROM operation. 3. Exceptions occur when EXCEPTION clears subtransactions. (For a stored procedure that contains the EXCEPTION statement, an implicit subtransaction is started when the statement is executed. After the statement is executed, the subtransaction is automatically cleared. During the clearing, exceptions may occur due to memory control.)

Example: Exceptions with **UPDATE**/**INSERT**

This example uses exception handling to perform either **UPDATE** or **INSERT**, as appropriate:

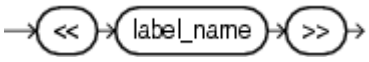
```
gaussdb=# CREATE TABLE db (a INT, b TEXT);
CREATE TABLE

gaussdb=# CREATE FUNCTION merge_db(key INT, data TEXT) RETURNS VOID AS
$$
BEGIN
    LOOP

-- First try to update the key
        UPDATE db SET b = data WHERE a = key;
        IF found THEN
            RETURN;
        END IF;
-- The key does not exist. Therefore, try to insert one. If someone else inserts the same key concurrently, the
unique key may fail to be obtained.
        BEGIN
            INSERT INTO db(a,b) VALUES (key, data);
            RETURN;
        EXCEPTION WHEN unique_violation THEN
        -- Do nothing, and loop to try the UPDATE again.
        END;
    END LOOP;
END;
$$
LANGUAGE plpgsql;
CREATE FUNCTION

gaussdb=# SELECT merge_db(1, 'david');
 merge_db
----------

(1 row)

gaussdb=# SELECT merge_db(1, 'dennis');
 merge_db
----------

(1 row)

--Delete FUNCTION and TABLE:
gaussdb=# DROP FUNCTION merge_db;
DROP FUNCTION

gaussdb=# DROP TABLE db;
DROP TABLE
```


10.8.7 GOTO Statements

A GOTO statement unconditionally transfers the control from the current statement to a labeled statement. The GOTO statement changes the execution logic. Therefore, use this statement only when necessary. Alternatively, you can use the EXCEPTION statement to handle issues in special scenarios. To execute a GOTO statement, the labeled statement must be unique.

Syntax

label declaration ::=



goto statement ::=

Examples

```
gaussdb=# CREATE OR REPLACE PROCEDURE GOTO_test()
AS
DECLARE
    v1  int;
BEGIN
    v1  := 0;
        LOOP
        EXIT WHEN v1 > 100;
                v1 := v1 + 2;
                if v1 > 25 THEN
                        GOTO pos1;
                END IF;
        END LOOP;
<<pos1>>
v1 := v1 + 10;
raise info 'v1 is %. ', v1;
END;
/
CREATE PROCEDURE

gaussdb=#call GOTO_test();
INFO:  v1 is 36.
 goto_test
-----------

(1 row)
```

Constraints

Using GOTO statements has the following constraints:

- Does not allow multiple labeled statements even if the statements are in different blocks.

  ```
  BEGIN
    GOTO pos1;
    <<pos1>>
    SELECT * FROM ...
    <<pos1>>
    UPDATE t1 SET ...
  END;
  ```


- Cannot transfer control to the IF, CASE, or LOOP statement.

```
BEGIN
  GOTO pos1;
  IF valid THEN
    <<pos1>>
    SELECT * FROM ...
  END IF;
END;
```

- Cannot transfer control from one IF clause to another, or from one WHEN clause in the CASE statement to another.

```
BEGIN
  IF valid THEN
    GOTO pos1;
    SELECT * FROM ...
  ELSE
    <<pos1>>
    UPDATE t1 SET ...
  END IF;
END;
```

- Cannot transfer control from an outer block to an inner BEGIN-END block.

```
BEGIN
  GOTO pos1;
  BEGIN
    <<pos1>>
    UPDATE t1 SET ...
  END;
END;
```

- Cannot transfer control from an exception handler to the current BEGIN-END block, but can transfer control to the upper-layer BEGIN-END block.

```
BEGIN
  <<pos1>>
  UPDATE t1 SET ...
  EXCEPTION
    WHEN condition THEN
      GOTO pos1;
END;
```

- To branch to a position that does not have an executable statement, you need to add the NULL statement.

```
DECLARE
  done  BOOLEAN;
BEGIN
  FOR i IN 1..50 LOOP
    IF done THEN
      GOTO end_loop;
    END IF;
    <<end_loop>>  -- You cannot run the GOTO statement to go here unless there is an executable
statement following it.
    NULL; -- The statement added here is used to avoid errors.
  END LOOP;  -- If the previous sentence NULL does not exist, an error is reported.
END;
/
```

10.9 Transaction Management

Calling a stored procedure automatically starts a transaction. When the calling is complete, the transaction is automatically committed, or rolled back upon an exception. In addition to automatic transaction control, you can also use COMMIT/ROLLBACK to control transactions in stored procedures. Running the COMMIT/ROLLBACK commands in a stored procedure will commit or roll back the current transaction and automatically starts a new transaction. All subsequent operations will be performed in the new transaction.

A savepoint is a special mark inside a transaction. It allows all commands that are executed after it was established to be rolled back, restoring the transaction state to what it was at the time of the savepoint. In a stored procedure, you can use savepoints to manage transactions. Currently, you can create, roll back, and release savepoints. If a savepoint for rollback is used in a stored procedure, only the modification of the current transaction is rolled back. The execution process of the stored procedure is not changed, and the values of local variables in the stored procedure are not rolled back.

Syntax

```
Define a savepoint.
   SAVEPOINT savepoint_name;
Roll back a savepoint.
   ROLLBACK TO [SAVEPOINT] savepoint_name;
Release a savepoint.
   RELEASE [SAVEPOINT] savepoint_name;
```

Usage Scenarios

The applicable contexts are as follows:

- COMMIT, ROLLBACK, and SAVEPOINT can be used in PL/SQL stored procedures.
- COMMIT, ROLLBACK, and SAVEPOINT can be used in stored procedures that contain EXCEPTION.
- COMMIT, ROLLBACK, and SAVEPOINT can be used in EXCEPTION statements of stored procedures.
- A stored procedure that contains COMMIT, ROLLBACK, or SAVEPOINT (which means the stored procedure is controlled by BEGIN, START, or END) can be called in a transaction block.
- A stored procedure that contains savepoints can be invoked in a subtransaction. That is, an externally defined savepoint is used in the stored procedure to roll back the transaction to the savepoint defined outside the stored procedure.
- A stored procedure is visible to a savepoint defined in the stored procedure. That is, the modification of the transaction can be rolled back to the savepoint defined in the stored procedure.
- COMMIT, ROLLBACK, and SAVEPOINT, as well as IF, FOR, CURSOR LOOP, and WHILE, can be called in most PL/SQL contexts and statements.
- A stored procedure or function that contains COMMIT, ROLLBACK, or SAVEPOINT can be called in the return values and simple expression calculation of stored procedures.

The following content can be committed or rolled back:

- DDL statements after COMMIT/ROLLBACK can be committed or rolled back.
- DML statements after COMMIT/ROLLBACK can be committed.
- GUC parameters in stored procedures can be committed or rolled back.

Usage Restrictions

COMMIT and ROLLBACK cannot be used in the following contexts:

- COMMIT, ROLLBACK, and SAVEPOINT cannot be called in stored procedures other than PL/SQL, such as PL/Java and PL/Python.
- COMMIT, ROLLBACK, SAVEPOINT and stored procedures that contain COMMIT, ROLLBACK, or SAVEPOINT cannot be called in functions.
- After SAVEPOINT is called in a transaction block, stored procedures that contain COMMIT/ROLLBACK cannot be called.
- Stored procedures that contain COMMIT, ROLLBACK, or SAVEPOINT cannot be called in TRIGGER.
- COMMIT, ROLLBACK, and SAVEPOINT cannot be invoked in EXECUTE statements.
- Stored procedures that contain COMMIT, ROLLBACK, or SAVEPOINT cannot be called in CURSOR statements.
- Stored procedures that contain IMMUTABLE or SHIPPABLE cannot call COMMIT, ROLLBACK, SAVEPOINT or another stored procedure that contain COMMIT, ROLLBACK, or SAVEPOINT.
- Stored procedures that contain COMMIT, ROLLBACK, or SAVEPOINT cannot be called in SQL statements other than SELECT PROC and CALL PROC.
- COMMIT, ROLLBACK, or SAVEPOINT cannot be called in a stored procedure whose header contains GUC parameters.
- COMMIT, ROLLBACK, or SAVEPOINT cannot be called in expressions or CURSOR and EXECUTE statements.
- Savepoints defined outside a stored procedure cannot be released in the stored procedure.
- An autonomous transaction and a stored procedure transaction are two independent transactions that cannot use the savepoints defined in each other.

The following content cannot be committed or rolled back:

- Variables declared or imported in stored procedures cannot be committed or rolled back.
- In stored procedures, GUC parameters that take effect only after a restart cannot be committed or rolled back.

Examples

- Example 1: COMMIT/ROLLBACK can be used in PL/SQL stored procedures. Subsequent examples depend on this case.

```
gaussdb=# DROP TABLE IF EXISTS EXAMPLE1;
NOTICE:  table "example1" does not exist, skipping
DROP TABLE
gaussdb=# CREATE TABLE EXAMPLE1(COL1 INT);
CREATE TABLE
gaussdb=# CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE()
AS
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1(COL1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
```



```
END;
/
CREATE PROCEDURE
gaussdb=# CALL TRANSACTION_EXAMPLE();
 transaction_example
---------------------

(1 row)
```

- Example 2:

  COMMIT and ROLLBACK can be used in stored procedures that contain EXCEPTION.

  COMMIT and ROLLBACK can be used in EXCEPTION statements of stored procedures.

  DDL statements after COMMIT/ROLLBACK can be committed or rolled back.

```
gaussdb=# CREATE OR REPLACE PROCEDURE TEST_COMMIT_INSERT_EXCEPTION_ROLLBACK()
AS
BEGIN
 DROP TABLE IF EXISTS TEST_COMMIT;
 CREATE TABLE TEST_COMMIT(A INT, B INT);
 INSERT INTO TEST_COMMIT SELECT 1, 1;
 COMMIT;
       CREATE TABLE TEST_ROLLBACK(A INT, B INT);
 RAISE EXCEPTION 'RAISE EXCEPTION AFTER COMMIT';
EXCEPTION
    WHEN OTHERS THEN
 INSERT INTO TEST_COMMIT SELECT 2, 2;
 ROLLBACK;
END;
/
CREATE PROCEDURE
gaussdb=# CALL TEST_COMMIT_INSERT_EXCEPTION_ROLLBACK();
NOTICE:  table "test_commit" does not exist, skipping
CONTEXT:  SQL statement "DROP TABLE IF EXISTS TEST_COMMIT"
PL/pgSQL function test_commit_insert_exception_rollback() line 3 at SQL statement
 test_commit_insert_exception_rollback
---------------------------------------

(1 row)
```

- Example 3: A stored procedure that contains COMMIT or ROLLBACK (which means the stored procedure is controlled by BEGIN, START, or END) can be called in a transaction block.

```
gaussdb=# BEGIN;
    -- For the definition of TEST_COMMIT_INSERT_EXCEPTION_ROLLBACK, see example 2.
    CALL TEST_COMMIT_INSERT_EXCEPTION_ROLLBACK();
END;
 test_commit_insert_exception_rollback
---------------------------------------

(1 row)
COMMIT
```

- Example 4: COMMIT and ROLLBACK, as well as IF, FOR, CURSOR LOOP, and WHILE, can be called in most PL/SQL contexts and statements.

```
gaussdb=# CREATE OR REPLACE PROCEDURE TEST_COMMIT2()
IS
BEGIN
    DROP TABLE IF EXISTS TEST_COMMIT;
    CREATE TABLE TEST_COMMIT(A INT);
    FOR I IN REVERSE 3..0 LOOP
 INSERT INTO TEST_COMMIT SELECT I;
 COMMIT;
    END LOOP;
    FOR I IN REVERSE 2..4 LOOP
 UPDATE TEST_COMMIT SET A=I;
 COMMIT;
```



```
    END LOOP;
EXCEPTION
WHEN OTHERS THEN
 INSERT INTO TEST_COMMIT SELECT 4;
    COMMIT;
END;
/
CREATE PROCEDURE
gaussdb=# CALL TEST_COMMIT2();
 test_commit2
--------------

(1 row)
```

- Example 5: Return values and simple expression calculation of stored procedures are supported.

```
gaussdb=# CREATE OR REPLACE PROCEDURE exec_func3(RET_NUM OUT INT)
AS
BEGIN
    RET_NUM := 1+1;
COMMIT;
END;
/
CREATE PROCEDURE
gaussdb=# CALL exec_func3('');
 ret_num
---------
       2
(1 row)
gaussdb=# CREATE OR REPLACE PROCEDURE exec_func4(ADD_NUM IN INT)
AS
SUM_NUM INT;
BEGIN
SUM_NUM := ADD_NUM + exec_func3();
COMMIT;
END;
/
CREATE PROCEDURE
gaussdb=# CALL exec_func4(1);
 exec_func4
------------

(1 row)
```

- Example 6: GUC parameters in stored procedures can be rolled back to a commit.

```
gaussdb=# SET explain_perf_mode='normal';
SET
gaussdb=# SHOW explain_perf_mode;
 explain_perf_mode
-------------------
 normal
(1 row)
gaussdb=# SHOW enable_force_vector_engine;
 enable_force_vector_engine
----------------------------
 off
(1 row)
gaussdb=# CREATE OR REPLACE PROCEDURE GUC_ROLLBACK()
AS
BEGIN
    SET enable_force_vector_engine = on;
    COMMIT;
    SET explain_perf_mode TO pretty;
    ROLLBACK;
END;
/
CREATE PROCEDURE
gaussdb=# CALL GUC_ROLLBACK();
 guc_rollback
```



```
--------------

(1 row)
gaussdb=# SHOW explain_perf_mode;
 explain_perf_mode
-------------------
 normal
(1 row)
gaussdb=# SHOW enable_force_vector_engine;
 enable_force_vector_engine
----------------------------
 on
(1 row)
gaussdb=# SET enable_force_vector_engine = off;
SET
```

- Example 7: A TRIGGER stored procedure cannot contain COMMIT or ROLLBACK or call another stored procedure that contains COMMIT or ROLLBACK.

```
gaussdb=# CREATE OR REPLACE FUNCTION FUNCTION_TRI_EXAMPLE2() RETURN TRIGGER
AS
EXP INT;
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1(col1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
    SELECT COUNT(*) FROM EXAMPLE1 INTO EXP;
END;
/
CREATE FUNCTION
gaussdb=# CREATE TRIGGER TRIGGER_EXAMPLE AFTER DELETE ON EXAMPLE1
FOR EACH ROW EXECUTE PROCEDURE FUNCTION_TRI_EXAMPLE2();
CREATE TRIGGER
gaussdb=# DELETE FROM EXAMPLE1;
ERROR:  Can not commit/rollback if it's atomic is true: can not use commit rollback in Complex SQL
CONTEXT:  PL/pgSQL function function_tri_example2() line 7 at COMMIT
```

- Example 8: Stored procedures that contain IMMUTABLE or SHIPPABLE cannot call COMMIT, ROLLBACK, or another stored procedure that contains COMMIT or ROLLBACK.

```
gaussdb=# CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE1()
IMMUTABLE
AS
EXP INT;
BEGIN
    FOR i IN 0..20 LOOP
        SELECT COUNT(*) FROM EXAMPLE1 INTO EXP;
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
END;
/
CREATE PROCEDURE
gaussdb=#CALL TRANSACTION_EXAMPLE1();
ERROR:  Can not commit/rollback if it's atomic is true: commit/rollback/savepoint is not allowed in a
non-volatile function
CONTEXT:  PL/pgSQL function transaction_example1() line 7 at COMMIT
```

- Example 9: Variables declared or passed in stored procedures cannot be committed.


```
gaussdb=# CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE2(EXP_OUT OUT INT)
AS
EXP INT:=-1;
BEGIN
    EXP_OUT := 0;
    EXP := 0;
    COMMIT;
    DBE_OUTPUT.PRINT_LINE('EXP IS:'||EXP);
    DBE_OUTPUT.PRINT_LINE('EXP_OUT IS:'||EXP_OUT);
    EXP := 1;
    EXP_OUT := 1;
    ROLLBACK;
    DBE_OUTPUT.PRINT_LINE('EXP IS:'||EXP);
    DBE_OUTPUT.PRINT_LINE('EXP_OUT IS:'||EXP_OUT);
END;
/
CREATE PROCEDURE
gaussdb=# CALL TRANSACTION_EXAMPLE2(1);
EXP IS:0
EXP_OUT IS:0
EXP IS:1
EXP_OUT IS:1
 exp_out
---------
       1
(1 row)
```

- Example 10: Calling in SQL statements (other than Select Procedure) is not supported.

```
gaussdb=# CREATE OR REPLACE FUNCTION TRANSACTION_EXAMPLE3()
RETURN INT
IS
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1 (col1) VALUES (i);
        IF i % 2 = 0 THEN
            EXECUTE IMMEDIATE 'COMMIT';
        ELSE
            EXECUTE IMMEDIATE 'ROLLBACK';
        END IF;
    END LOOP;
    RETURN 1;
END;
/
CREATE PROCEDURE
gaussdb=# SELECT * FROM example1 WHERE col1=TRANSACTION_EXAMPLE3();
ERROR:  cannot call transaction statements in EXECUTE IMMEDIATE statement.
CONTEXT:  PL/pgSQL function transaction_example3() line 6 at EXECUTE statement
```

- Example 11: COMMIT or ROLLBACK cannot be called in a stored procedure whose header contains GUC parameters.

```
gaussdb=# CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE4()
SET ARRAY_NULLS TO "ON"
AS
BEGIN
    FOR i IN 0..20 LOOP
        INSERT INTO EXAMPLE1 (col1) VALUES (i);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
    END LOOP;
END;
/
CREATE PROCEDURE
gaussdb=# CALL TRANSACTION_EXAMPLE4();
ERROR:  Can not commit/rollback if it's atomic is true: transaction statement in store procedure with GUC setting in option clause is not supported
CONTEXT:  PL/pgSQL function transaction_example4() line 6 at COMMIT
```


- Example 12: A stored procedure whose cursor is open cannot contain COMMIT or ROLLBACK.

```
gaussdb=# CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE5(INTIN IN INT, INTOUT OUT
INT)
AS
BEGIN
INTOUT := INTIN + 1;
COMMIT;
END;
/
CREATE PROCEDURE
gaussdb=# CREATE OR REPLACE PROCEDURE TRANSACTION_EXAMPLE6()
AS
CURSOR CURSOR1(EXPIN INT)
IS SELECT TRANSACTION_EXAMPLE5(EXPIN);
INTEXP INT;
BEGIN
    FOR i IN 0..20 LOOP
        OPEN CURSOR1(i);
        FETCH CURSOR1 INTO INTEXP;
        INSERT INTO EXAMPLE1(COL1) VALUES (INTEXP);
        IF i % 2 = 0 THEN
            COMMIT;
        ELSE
            ROLLBACK;
        END IF;
        CLOSE CURSOR1;
    END LOOP;
END;
/
CREATE PROCEDURE
gaussdb=# CALL TRANSACTION_EXAMPLE6();
ERROR:  Can not commit/rollback if it's atomic is true: transaction statement in store procedure used
as cursor is not supported
CONTEXT:  PL/pgSQL function transaction_example5(integer) line 4 at COMMIT
referenced column: transaction_example5
PL/pgSQL function transaction_example6() line 8 at FETCH
```

- Example 13: COMMIT or ROLLBACK cannot be called in expressions or CURSOR and EXECUTE statements.

```
gaussdb=# CREATE OR REPLACE PROCEDURE exec_func1()
AS
BEGIN
    CREATE TABLE TEST_exec(A INT);
COMMIT;
END;
/
CREATE PROCEDURE
gaussdb=# CREATE OR REPLACE PROCEDURE exec_func2()
AS
BEGIN
EXECUTE exec_func1();
COMMIT;
END;
/
CREATE PROCEDURE
gaussdb=# CALL exec_func2();
ERROR:  Can not commit/rollback if it's atomic is true: transaction statement in store procedure used
as a expression is not supported
CONTEXT:  PL/pgSQL function exec_func1() line 4 at COMMIT
PL/pgSQL function exec_func2() line 3 at EXECUTE statement
```

- Example 14: Roll back some modifications of stored procedure on a transaction to a savepoint.

```
gaussdb=# CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE1()
AS
BEGIN
    INSERT INTO EXAMPLE1 VALUES(1);
    SAVEPOINT s1;
```



```
    INSERT INTO EXAMPLE1 VALUES(2);
    ROLLBACK TO s1;  -- Roll back the insertion of record 2.
    INSERT INTO EXAMPLE1 VALUES(3);
END;
/
CREATE PROCEDURE
gaussdb=# CALL STP_SAVEPOINT_EXAMPLE1();
 stp_savepoint_example1
------------------------

(1 row)
```

- Example 15: Roll back a stored procedure to a savepoint defined outside the stored procedure.

```
gaussdb=# TRUNCATE TABLE EXAMPLE1;
TRUNCATE TABLE
gaussdb=# CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE2()
AS
BEGIN
    INSERT INTO EXAMPLE1 VALUES(2);
    ROLLBACK TO s1;  -- Roll back the insertion of record 2.
    INSERT INTO EXAMPLE1 VALUES(3);
END;
/
CREATE PROCEDURE
gaussdb=# BEGIN;
BEGIN
gaussdb=# INSERT INTO EXAMPLE1 VALUES(1);
INSERT 0 1
gaussdb=# SAVEPOINT s1;
SAVEPOINT
gaussdb=# CALL STP_SAVEPOINT_EXAMPLE2();
 stp_savepoint_example2
------------------------

(1 row)
gaussdb=# SELECT * FROM EXAMPLE1;
 col1
------
    1
    3
(2 rows)
gaussdb=# COMMIT;
COMMIT
```

- Example 16: Savepoints defined outside the stored procedure cannot be released in the stored procedure.

```
gaussdb=# CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE3()
AS
BEGIN
    INSERT INTO EXAMPLE1 VALUES(2);
    RELEASE SAVEPOINT s1; -- Release the savepoint defined outside the stored procedure.
    INSERT INTO EXAMPLE1 VALUES(3);
END;
/
CREATE PROCEDURE
gaussdb=# BEGIN;
BEGIN
gaussdb=# INSERT INTO EXAMPLE1 VALUES(1);
INSERT 0 1
gaussdb=# SAVEPOINT s1;
SAVEPOINT
gaussdb=# CALL STP_SAVEPOINT_EXAMPLE3();
ERROR:  cannot release outer savepoint
CONTEXT:  PL/pgSQL function stp_savepoint_example3() line 4 at RELEASE SAVEPOINT
gaussdb=# COMMIT;
ROLLBACK
```

- Example 17: Roll back an external SQL or other stored procedure to a savepoint defined in the stored procedure.


```
gaussdb=# TRUNCATE TABLE EXAMPLE1;
TRUNCATE TABLE
gaussdb=# CREATE OR REPLACE PROCEDURE STP_SAVEPOINT_EXAMPLE4()
AS
BEGIN
    INSERT INTO EXAMPLE1 VALUES(1);
    SAVEPOINT s1;
    INSERT INTO EXAMPLE1 VALUES(2);
END;
/
CREATE PROCEDURE
gaussdb=# BEGIN;
BEGIN
gaussdb=# INSERT INTO EXAMPLE1 VALUES(3);
INSERT 0 1
gaussdb=# CALL STP_SAVEPOINT_EXAMPLE4();
 stp_savepoint_example4
------------------------

(1 row)
gaussdb=# ROLLBACK TO SAVEPOINT s1; -- Roll back the insertion of record 2 to the stored
procedure.
ROLLBACK
gaussdb=# SELECT * FROM EXAMPLE1;
 col1
------
    3
    1
(2 rows)
gaussdb=# COMMIT;
COMMIT
```

10.10 Other Statements

10.10.1 Lock Operations

GaussDB provides multiple lock modes to control concurrent accesses to table data. These modes are used when Multi-Version Concurrency Control (MVCC) cannot give expected behaviors. Alike, most GaussDB commands automatically apply appropriate locks to ensure that called tables are not deleted or modified in an incompatible manner during command execution. For example, when concurrent operations exist, ALTER TABLE cannot be executed on the same table.

10.10.2 Cursor Operations

GaussDB provides cursors as a data buffer for users to store execution results of SQL statements. Each cursor region has a name. Users can use SQL statements to obtain records one by one from cursors and grant the records to master variables, then being processed further by host languages.

Cursor operations include cursor definition, open, fetch, and close operations.

For the complete example of cursor operations, see **Explicit Cursor**.

10.11 Cursors

10.11.1 Overview

To process SQL statements, the stored procedure process assigns a memory segment to store context association. Cursors are handles or pointers pointing to context regions. With cursors, stored procedures can control alterations in context regions.

NOTICE

- When a stored procedure uses returns to return a cursor, if JDBC is used to call the stored procedure, the returned cursor is unavailable. Using **out** to output parameters is not affected.
- When a stored procedure contains COMMIT/ROLLBACK, an explicit cursor caches all data of the cursor to ensure that the cursor is still available after COMMIT/ROLLBACK. If the cursor data volume is large, this process may take a long time.
- After table data is modified in a stored procedure, the cursor related to the table is started, and the data in the FETCH cursor continues after rollback. In this case, an error is reported.

Cursors are classified into explicit cursors and implicit cursors. **Table 10-2** shows the usage conditions of explicit and implicit cursors for different SQL statements.

Table 10-2 Cursor usage conditions

SQL Statement	Cursor
Non-query statements	Implicit
Query statements with single-line results	Implicit or explicit
Query statements with multi-line results	Explicit

10.11.2 Explicit Cursor

An explicit cursor is used to process query statements, particularly when query results are multiple records.

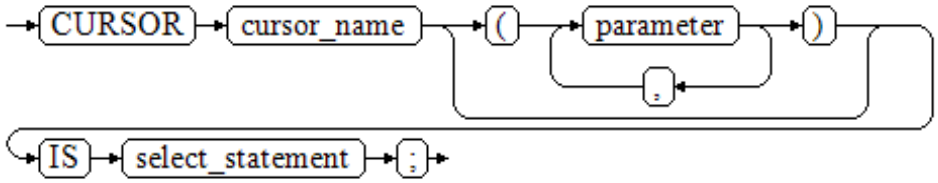
Procedure

An explicit cursor performs the following six PL/SQL steps to process query statements:

Step 1 Define a static cursor: Define a cursor name and its corresponding SELECT statement.

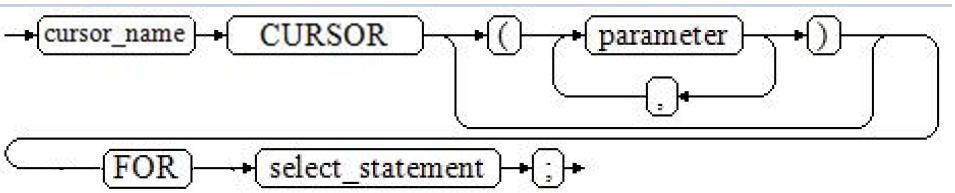
Figure 10-27 shows the syntax diagram for defining a static cursor.

Figure 10-27 static_cursor_define::=



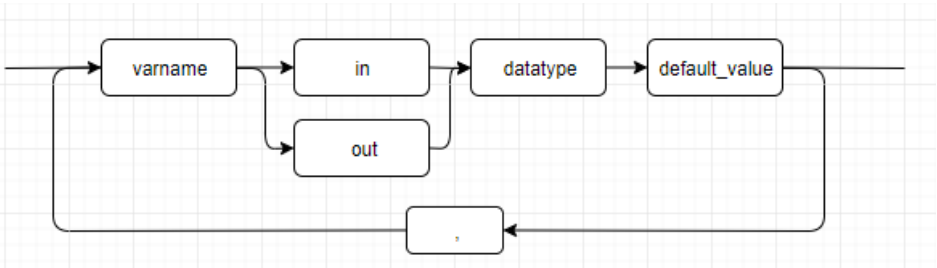
or

Figure 10-28 static_cursor_define::=



Parameter description:

- *cursor_name*: defines a cursor name.
- **parameter**: cursor parameter, which can only be an input parameter. The default value can be defined by **:=**, **=**, or **default**. The format is as follows.



- *select_statement*: specifies a query statement.

NOTE

- The system automatically determines whether the cursor can be used for backward fetching based on the execution plan.
- In syntax, **parameter** can be an output parameter, but its behavior is the same as that of an input parameter.

Define a dynamic cursor: Define a **ref** cursor, which means that the cursor can be opened dynamically by a set of static SQL statements. Define the type of the **ref** cursor first, and then the cursor variable of this cursor type. Dynamically bind a SELECT statement through OPEN FOR when the cursor is opened.

Figure 10-29 and **Figure 10-30** show the syntax diagrams for defining a dynamic cursor.

Figure 10-29 cursor_typename::=

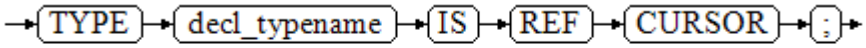
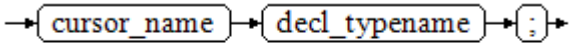


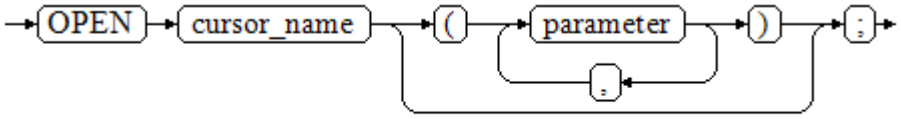
Figure 10-30 dynamic_cursor_define::=



Step 2 Open the static cursor: Execute the SELECT statement corresponding to the cursor. The query result is placed in the workspace and the pointer directs to the head of the workspace to identify the cursor result set. If the cursor query statement carries the **FOR UPDATE** option, the OPEN statement locks the data rows corresponding to the cursor result set in the database table.

Figure 10-31 shows the syntax diagram for opening a static cursor.

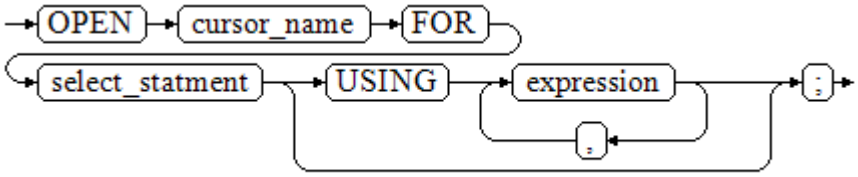
Figure 10-31 open_static_cursor::=



Open the dynamic cursor: Use the OPEN FOR statement to open the dynamic cursor and the SQL statement is dynamically bound.

Figure 10-32 shows the syntax diagrams for opening a dynamic cursor.

Figure 10-32 open_dynamic_cursor::=

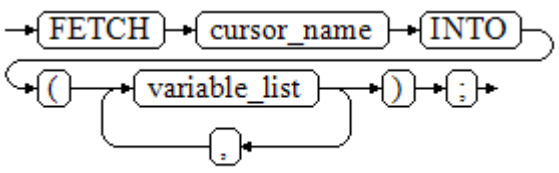


A PL/SQL program cannot use the OPEN statement to repeatedly open a cursor.

Step 3 Fetch cursor data: Retrieve data rows in the result set and place them in specified output variables.

Figure 10-33 shows the syntax diagrams for fetching cursor data.

Figure 10-33 fetch_cursor::=



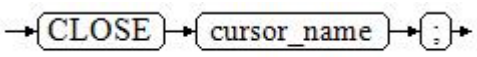
Step 4 Process the record.

Step 5 Continue to process until the active set has no record.

Step 6 Close the cursor: After you fetch and process the data in the cursor result set, close the cursor in time to release system resources used by the cursor and invalidate the workspace of the cursor so that the FETCH statement cannot be used to fetch data anymore. A closed cursor can be reopened by an OPEN statement.

Figure 10-34 shows the syntax diagram for closing a cursor.

Figure 10-34 close_cursor::=



----End

Attributes

Cursor attributes are used to control program procedures or know program status. When a DML statement is executed, the PL/SQL opens a built-in cursor and processes its result. A cursor is a memory segment for maintaining query results. It is opened when a DML statement is executed and closed when the execution is finished. An explicit cursor has the following attributes:

- **%FOUND**: Boolean attribute, which returns **TRUE** if the last fetch returns a row.
- **%NOTFOUND**: Boolean attribute, which returns **TRUE** if the last fetch fails to return a row.
- **%ISOPEN**: Boolean attribute, which returns **TRUE** if the cursor has been opened.
- **%ROWCOUNT**: numeric attribute, which returns the number of records fetched from the cursor.

Examples

DDL and DML statements are prepared. Subsequent examples in this section depend on this case.

```
gaussdb=# DROP SCHEMA IF EXISTS hr CASCADE;
NOTICE:  schema "hr" does not exist, skipping
DROP SCHEMA
gaussdb=# CREATE SCHEMA hr;
CREATE SCHEMA
gaussdb=# SET current_schema = hr;
SET
gaussdb=# DROP TABLE IF EXISTS sections;
NOTICE:  table "sections" does not exist, skipping
DROP TABLE
gaussdb=# DROP TABLE IF EXISTS staffs;
NOTICE:  table "staffs" does not exist, skipping
DROP TABLE
gaussdb=# DROP TABLE IF EXISTS department;
NOTICE:  table "department" does not exist, skipping
DROP TABLE

-- Create a department table.
gaussdb=# CREATE TABLE sections(
```



```
    section_name varchar(100),
    place_id     int,
    section_id   int
);
CREATE TABLE
gaussdb=# INSERT INTO sections VALUES ('hr',1,1);
INSERT 0 1
-- Create an employee table.
gaussdb=# CREATE TABLE staffs(
    staff_id number(6),
    salary number(8,2),
    section_id int,
    first_name varchar(20)
);
CREATE TABLE
gaussdb=# INSERT INTO staffs VALUES (1,100,1,'Tom');
INSERT 0 1
-- Create a department table.
gaussdb=# CREATE TABLE department(
    section_id int
);
CREATE TABLE
-- Specify the method for passing cursor parameters.
gaussdb=# CREATE OR REPLACE PROCEDURE cursor_proc1()
AS
DECLARE
    DEPT_NAME VARCHAR(100);
    DEPT_LOC NUMBER(4);
    -- Define a cursor.
    CURSOR C1 IS
        SELECT section_name, place_id FROM hr.sections WHERE section_id <= 50;
    CURSOR C2(sect_id INTEGER) IS
        SELECT section_name, place_id FROM hr.sections WHERE section_id <= sect_id;
    TYPE CURSOR_TYPE IS REF CURSOR;
    C3 CURSOR_TYPE;
    SQL_STR VARCHAR(100);
BEGIN
    OPEN C1;-- Open the cursor.
    LOOP
        -- Fetch data from the cursor.
        FETCH C1 INTO DEPT_NAME, DEPT_LOC;
        EXIT WHEN C1%NOTFOUND;
        DBE_OUTPUT.PRINT_LINE(DEPT_NAME||'---'||DEPT_LOC);
    END LOOP;
    CLOSE C1;-- Close the cursor.

    OPEN C2(10);
    LOOP
        FETCH C2 INTO DEPT_NAME, DEPT_LOC;
        EXIT WHEN C2%NOTFOUND;
        DBE_OUTPUT.PRINT_LINE(DEPT_NAME||'---'||DEPT_LOC);
    END LOOP;
    CLOSE C2;

    SQL_STR := 'SELECT section_name, place_id FROM hr.sections WHERE section_id <= :DEPT_NO;';
    OPEN C3 FOR SQL_STR USING 50;
    LOOP
        FETCH C3 INTO DEPT_NAME, DEPT_LOC;
        EXIT WHEN C3%NOTFOUND;
        DBE_OUTPUT.PRINT_LINE(DEPT_NAME||'---'||DEPT_LOC);
    END LOOP;
    CLOSE C3;
END;
/
CREATE PROCEDURE
gaussdb=# CALL cursor_proc1();
hr---1
hr---1
hr---1
```



```
 cursor_proc1
--------------

(1 row)

gaussdb=#DROP PROCEDURE cursor_proc1;
DROP PROCEDURE
-- Give a salary raise to employees whose salary is lower than 3000 by adding 500.
gaussdb=# CREATE TABLE hr.staffs_t1 AS TABLE hr.staffs;
INSERT 0 1
gaussdb=# CREATE OR REPLACE PROCEDURE cursor_proc2()
AS
DECLARE
  V_EMPNO  NUMBER(6);
  V_SAL    NUMBER(8,2);
  CURSOR C IS SELECT staff_id, salary FROM hr.staffs_t1;
BEGIN
  OPEN C;
  LOOP
    FETCH C INTO V_EMPNO, V_SAL;
    EXIT WHEN C%NOTFOUND;
    IF V_SAL<=3000 THEN
         UPDATE hr.staffs_t1 SET salary =salary + 500 WHERE staff_id = V_EMPNO;
    END IF;
  END LOOP;
  CLOSE C;
END;
/
CREATE PROCEDURE
gaussdb=# CALL cursor_proc2();
 cursor_proc2
--------------

(1 row)
-- Delete the stored procedure.
gaussdb=# DROP PROCEDURE cursor_proc2;
DROP PROCEDURE
gaussdb=# DROP TABLE hr.staffs_t1;
DROP TABLE
-- Use function parameters of the SYS_REFCURSOR type.
gaussdb=# CREATE OR REPLACE PROCEDURE proc_sys_ref(O OUT SYS_REFCURSOR)
IS
C1 SYS_REFCURSOR;
BEGIN
OPEN C1 FOR SELECT section_id FROM hr.sections ORDER BY section_id;
O := C1;
END;
/
CREATE PROCEDURE
gaussdb=# DECLARE
C1 SYS_REFCURSOR;
TEMP NUMBER(4);
BEGIN
proc_sys_ref(C1);
LOOP
  FETCH C1 INTO TEMP;
  DBE_OUTPUT.PRINT_LINE(C1%ROWCOUNT);
  EXIT WHEN C1%NOTFOUND;
END LOOP;
END;
/
1
1
ANONYMOUS BLOCK EXECUTE

-- Delete the stored procedure.
gaussdb=# DROP PROCEDURE proc_sys_ref;
DROP PROCEDURE
```


10.11.3 Implicit Cursor

Implicit cursors are automatically set by the system for non-query statements such as modify or delete operations, along with their workspace. Implicit cursors are named **SQL**, which is defined by the system.

Overview

Implicit cursor operations, such as definition, open, value-grant, and close operations, are automatically performed by the system and do not need users to process. Users can use only attributes related to implicit cursors to complete operations. In workspace of implicit cursors, the data of the latest SQL statement is stored and is not related to explicit cursors defined by users.

Format call: **SQL%**

NOTE

- INSERT, UPDATE, DELETE, and SELECT statements do not need defined cursors.
- In A-compatible mode, if the GUC parameter **behavior_compat_options** is set to **compat_cursor**, implicit cursors are valid across stored procedures.
- After SMP-related GUC parameters are enabled (the GUC parameter **query_dop** is set to a value greater than 1 and **plsql_beta_feature** is set to **enable_plsql_smp**), query statements that do not involve INSERT, UPDATE, and DELETE in cursors can be executed on SMP.
- Implicit cursor attributes are not affected by the commit\rollback operation.

Attributes

An implicit cursor has the following attributes:

- **SQL%FOUND**: Boolean attribute, which returns **TRUE** if the last fetch returns a row.
- **SQL%NOTFOUND**: Boolean attribute, which returns **TRUE** if the last fetch fails to return a row.
- **SQL%ROWCOUNT**: numeric attribute, which returns the number of records fetched from the cursor.
- **SQL%ISOPEN**: Boolean attribute, whose value is always **FALSE**. Close implicit cursors immediately after an SQL statement is run.

Examples

```
-- Delete all employees in a department from the hr.staffs table. If the department has no employees, delete the department from the hr.department table.
gaussdb=# CREATE OR REPLACE PROCEDURE proc_cursor3()
AS
   DECLARE
   V_DEPTNO NUMBER(4) := 100;
   BEGIN
      DELETE FROM hr.staffs WHERE section_id = V_DEPTNO;
      -- Proceed based on cursor status.
      IF SQL%NOTFOUND THEN
      DELETE FROM hr.department WHERE section_id = V_DEPTNO;
      END IF;
   END;
/
CREATE PROCEDURE
```



```
gaussdb=# CALL proc_cursor3();
 proc_cursor3
--------------

(1 row)

-- Delete the stored procedure and the temporary table.
gaussdb=# DROP PROCEDURE proc_cursor3;
DROP PROCEDURE
-- If SELECT implicit cursors need to be executed concurrently, enable the following GUC parameters:
gaussdb=# SET query_dop=4;
SET
gaussdb=# SET sql_beta_feature= 'enable_plsql_smp';

-- Select the name of the employee whose ID is 1 from the employee table hr.staffs.
gaussdb=# CREATE OR REPLACE PROCEDURE prc_cursor_smp()
AS
    name varchar(20);
BEGIN
   SELECT first_name FROM hr.staffs WHERE staff_id = 1 INTO name;
   dbe_output.print_line('result is: '|| name);
END;
/
CREATE PROCEDURE
-- Execute the stored procedure.
gaussdb=# CALL prc_cursor_smp();
result is: Tom
 prc_cursor_smp
----------------

(1 row)
```

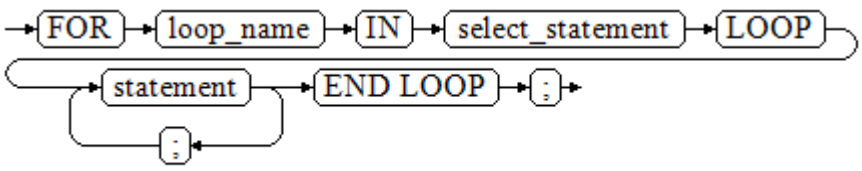
10.11.4 Cursor Loop

Use of cursors in WHILE and LOOP statements is called a cursor loop. Generally, OPEN, FETCH, and CLOSE statements are called in this kind of loop. The following describes a loop that simplifies a cursor loop without the need for these operations. This mode is applicable to a static cursor loop, without executing four steps about a static cursor.

Syntax

Figure 10-35 shows the syntax diagram of the FOR AS loop.

Figure 10-35 FOR_AS_loop::=



Precautions

- The UPDATE operation for the queried table is not allowed in the loop statement.
- The variable *loop_name* is automatically defined and is valid only in this loop. Its type is the same as that in the query result of *select_statement*. The value of *loop_name* is the query result of *select_statement*.

- The specific type of the *loop_name* variable is not parsed during compilation. If the specific type needs to be parsed (for example, *loop_name* is used as the input and output parameters of an overloaded function or stored procedure), a compilation error is reported. To parse the specific type of a variable, set **behavior_compat_options** to **allow_procedure_compile_check**.
- The **%FOUND**, **%NOTFOUND**, and **%ROWCOUNT** attributes access the same internal variable in GaussDB. Transactions and the anonymous block do not support multiple cursor accesses at the same time.
- You can enable the SMP-related GUC parameter (set the GUC parameter **query_dop** to a value greater than 1 and **plsql_beta_feature** to '**enable_plsql_smp**') so that the cursor loop can be executed on the SMP.

Examples

```
-- If cursors need to be executed cyclically and concurrently, enable the following GUC parameters:
gaussdb=# SET query_dop=4;
SET
gaussdb=# SET sql_beta_feature='enable_plsql_smp';
SET
gaussdb=# BEGIN
FOR ROW_TRANS IN
        SELECT first_name FROM hr.staffs
    LOOP
        DBE_OUTPUT.PRINT_LINE (ROW_TRANS.first_name );
    END LOOP;
END;
/
Tom
ANONYMOUS BLOCK EXECUTE

-- Create a table.
gaussdb=# CREATE TABLE integerTable1( A INTEGER);
CREATE TABLE
gaussdb=# CREATE TABLE integerTable2( B INTEGER);
CREATE TABLE
gaussdb=# INSERT INTO integerTable2 VALUES(2);
INSERT 0 1
-- Multiple cursors share the parameters of cursor attributes.
gaussdb=# DECLARE
   CURSOR C1 IS SELECT A FROM integerTable1;-- Declare the cursor.
   CURSOR C2 IS SELECT B FROM integerTable2;
   PI_A INTEGER;
   PI_B INTEGER;
BEGIN
    OPEN C1;-- Open the cursor.
   OPEN C2;
   FETCH C1 INTO PI_A; ---- The values of C1%FOUND and C2%FOUND are FALSE.
   FETCH C2 INTO PI_B; ---- The values of C1%FOUND and C2%FOUND are TRUE.
   -- Determine the cursor status.
   IF C1%FOUND THEN
      IF C2%FOUND THEN
        DBE_OUTPUT.PRINT_LINE('Dual cursor share parameter.');
      END IF;
   END IF;
    CLOSE C1;-- Close the cursor.
   CLOSE C2;
END;
/
ANONYMOUS BLOCK EXECUTE

-- Delete the temporary table.
gaussdb=# DROP TABLE integerTable1;
DROP TABLE
gaussdb=# DROP TABLE integerTable2;
DROP TABLE
```


10.12 Advanced Packages

Advanced packages have two sets of interfaces. The first set is basic interfaces, and the second set is secondary encapsulation interfaces that are used improve usability. The second set is recommended.

10.12.1 Basic Interfaces

10.12.1.1 PKG_SERVICE

Table 10-3 lists all APIs supported by PKG_SERVICE.

Table 10-3 PKG_SERVICE

API	Description
PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE	Checks whether a context is registered.
PKG_SERVICE.SQL_CLEAN_ALL_CONTEXTS	Deregisters all registered contexts.
PKG_SERVICE.SQL_REGISTER_CONTEXT	Registers a context.
PKG_SERVICE.SQL_UNREGISTER_CONTEXT	Deregisters a context.
PKG_SERVICE.SQL_SET_SQL	Sets an SQL statement for a context. Currently, only the SELECT statement is supported.
PKG_SERVICE.SQL_RUN	Executes the configured SQL statement on a context.
PKG_SERVICE.SQL_NEXT_ROW	Reads the next row of data in a context.
PKG_SERVICE.SQL_GET_VALUE	Reads a dynamically defined column value in a context.
PKG_SERVICE.SQL_SET_RESULT_TYPE	Dynamically defines a column of a context based on the type OID.
PKG_SERVICE.JOB_CANCEL	Removes a scheduled task by task ID.
PKG_SERVICE.JOB_FINISH	Disables or enables scheduled task execution.
PKG_SERVICE.JOB_SUBMIT	Submits a scheduled task. Job ID can be automatically generated by the system or specified manually.

API	Description
PKG_SERVICE.JOB_UPDATE	Modifies user-definable attributes of a scheduled task, including the task content, next-execution time, and execution interval.
PKG_SERVICE.SUBMIT_ON_NODES	Submits a task to all nodes. The task ID is automatically generated by the system.
PKG_SERVICE.ISUBMIT_ON_NODES	Submits a job to all nodes. The job ID is specified by the user.
PKG_SERVICE.SQL_GET_ARRAY_RESULT	Obtains the array value returned in the context.
PKG_SERVICE.SQL_GET_VARIABLE_RESULT	Obtains the column value returned in the context.

- PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE

  This function checks whether a context is registered. This function transfers the ID of the context to be queried. If the context exists, **TRUE** is returned. Otherwise, **FALSE** is returned.

  The prototype of the PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE function is as follows:

  ```
  PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE(
   context_id      IN INTEGER
  )
  RETURN BOOLEAN;
  ```

  Table 10-4 PKG_SERVICE.SQL_IS_CONTEXT_ACTIVE parameters

Parameter	Description
context_id	ID of the context to be queried

- PKG_SERVICE.SQL_CLEAN_ALL_CONTEXTS

  This function cancels all contexts.

  The prototype of the PKG_SERVICE.SQL_CLEAN_ALL_CONTEXTS function is as follows:

  ```
  PKG_SERVICE.SQL_CLEAN_ALL_CONTEXTS(
  )
  RETURN VOID;
  ```

- PKG_SERVICE.SQL_REGISTER_CONTEXT

  This function opens a context, which is the prerequisite for the subsequent operations in the context. This function does not transfer any parameter. It automatically generates context IDs in an ascending order and returns values to integer variables.

  The prototype of the PKG_SERVICE.SQL_REGISTER_CONTEXT function is as follows:


```
DBE_SQL.REGISTER_CONTEXT(
)
RETURN INTEGER;
```

- PKG_SERVICE.SQL_UNREGISTER_CONTEXT

  This function closes a context, which is the end of each operation in the context. If this function is not called when the stored procedure ends, the memory is still occupied by the context. Therefore, remember to close a context when you do not need to use it. If an exception occurs, the stored procedure exits but the context is not closed. Therefore, you are advised to include this API in the exception handling of the stored procedure.

  The prototype of the PKG_SERVICE.SQL_UNREGISTER_CONTEXT function is as follows:

```
PKG_SERVICE.SQL_UNREGISTER_CONTEXT(
 context_id     IN INTEGER
)
RETURN INTEGER;
```

Table 10-5 PKG_SERVICE.SQL_UNREGISTER_CONTEXT parameters

Parameter	Description
context_id	ID of the context to be closed.

- PKG_SERVICE.SQL_SET_SQL

  This function parses the query statement of a given context. The input query statement is executed immediately. Currently, only the SELECT query statement can be parsed. The statement parameters can be transferred only through the TEXT type. The length cannot exceed 1 GB.

  The prototype of the PKG_SERVICE.SQL_SET_SQL function is as follows:

```
PKG_SERVICE.SQL_SET_SQL(
context_id     IN INTEGER,
query_string   IN TEXT,
language_flag  IN INTEGER
)
RETURN BOOLEAN;
```

Table 10-6 PKG_SERVICE.SQL_SET_SQL parameters

Parameter	Description
context_id	ID of the context whose query statement is to be parsed.
query_string	Query statement to be parsed.
language_flag	Version language number, which specifies the behavior of different versions. • **1**: incompatible version. • **2**: version in A-compatible mode.

- PKG_SERVICE.SQL_RUN

This function executes a given context. It receives a context ID first, and the data obtained after execution is used for subsequent operations. Currently, only the SELECT query statement can be executed.

The prototype of the PKG_SERVICE.SQL_RUN function is as follows:

```
PKG_SERVICE.SQL_RUN(
context_id     IN INTEGER,
)
RETURN INTEGER;
```

Table 10-7 PKG_SERVICE.SQL_RUN parameters

Parameter	Description
context_id	ID of the context whose query statement is to be parsed.

- PKG_SERVICE.SQL_NEXT_ROW

  This function returns the number of data rows returned after the SQL statement is executed. Each time the API is executed, the system obtains a set of new rows until all data is read.

  The prototype of the PKG_SERVICE.SQL_NEXT_ROW function is as follows:

```
PKG_SERVICE.SQL_NEXT_ROW(
context_id     IN INTEGER,
)
RETURN INTEGER;
```

Table 10-8 PKG_SERVICE.SQL_NEXT_ROW parameters

Parameter	Description
context_id	ID of the context to be executed

- PKG_SERVICE.SQL_GET_VALUE

  This function returns the context element value in a specified position of a context and accesses the data obtained by PKG_SERVICE.SQL_NEXT_ROW.

  The prototype of the PKG_SERVICE.SQL_GET_VALUE function is as follows:

```
PKG_SERVICE.SQL_GET_VALUE(
context_id         IN    INTEGER,
pos                IN    INTEGER,
col_type           IN    ANYELEMENT
)
RETURN ANYELEMENT;
```

Table 10-9 PKG_SERVICE.SQL_GET_VALUE parameters

Parameter	Description
context_id	ID of the context to be executed
pos	Position of a dynamically defined column in the query
col_type	Variable of any type, which defines the return value type of columns.

- PKG_SERVICE.SQL_SET_RESULT_TYPE

  This function defines columns returned from a given context and can be used only for contexts defined by SELECT. The defined columns are identified by the relative positions in the query list. The prototype of PKG_SERVICE.SQL_SET_RESULT_TYPE is as follows:

```
PKG_SERVICE.SQL_SET_RESULT_TYPE(
context_id     IN INTEGER,
pos            IN INTEGER,
coltype_oid    IN ANYELEMENT,
maxsize        IN INTEGER
)
RETURN INTEGER;
```

  Table 10-10 PKG_SERVICE.SQL_SET_RESULT_TYPE parameters

Parameter	Description
context_id	ID of the context to be executed
pos	Position of a dynamically defined column in the query
coltype_oid	Variable of any type. The OID of the corresponding type can be obtained based on the variable type.
maxsize	Length of a defined column

- PKG_SERVICE.JOB_CANCEL

  The stored procedure CANCEL deletes a specified task.

  The prototype of the PKG_SERVICE.JOB_CANCEL function is as follows:

```
PKG_SERVICE.JOB_CANCEL(
id  IN  INTEGER);
```

  Table 10-11 PKG_SERVICE.JOB_CANCEL parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	integer	IN	No	Specifies the job ID.

- PKG_SERVICE.JOB_FINISH

  The stored procedure FINISH disables or enables a scheduled task.

  The prototype of the PKG_SERVICE.JOB_FINISH function is as follows:

```
PKG_SERVICE.JOB_FINISH(
id         IN   INTEGER,
broken     IN   BOOLEAN,
next_time  IN   TIMESTAMP  DEFAULT  sysdate);
```


Table 10-12 PKG_SERVICE.JOB_FINISH parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
id	integer	IN	No	Specifies the job ID.
broken	boolean	IN	No	Specifies the status flag, **true** for broken and **false** for not broken. The current job is updated based on the parameter value **true** or **false**. If the parameter is left empty, the job status remains unchanged.
next_time	timestamp	IN	Yes	Specifies the next execution time. The default value is the current system time. If **broken** is set to **true**, **next_time** is updated to **'4000-1-1'**. If **broken** is set to **false** and **next_time** is not empty, **next_time** is updated for the job. If **next_time** is empty, it will not be updated. This parameter can be omitted, and its default value will be used in this case.

- PKG_SERVICE.JOB_SUBMIT

  The stored procedure JOB_SUBMIT submits a scheduled task provided by the system.

  The prototype of the PKG_SERVICE.JOB_SUBMIT function is as follows:

```
PKG_SERVICE.JOB_SUBMIT(
id            IN   BIGINT,
content       IN   TEXT,
next_time     IN   TIMESTAMP DEFAULT sysdate,
interval_time IN   TEXT  DEFAULT 'null',
job           OUT  INTEGER);
```

  NOTE

  When a scheduled job is created, the system binds the current database and the username to the job by default. This function can be called by using **call** or **select**. If you call this function by using **select**, there is no need to specify output parameters. To call this function within a stored procedure, use **perform**. If the committed SQL statement task uses a non-public schema, specify the schema to a table schema or a function schema, or add **set current_schema = xxx;** before the SQL statement.

Table 10-13 PKG_SERVICE.JOB_SUBMIT parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
id	bigint	IN	No	Specifies the job ID. If the input ID is **NULL**, a job ID is generated internally.
content	text	IN	No	Specifies the SQL statement to be executed. One or multiple DMLs, anonymous blocks, and statements for calling stored procedures, or all three combined are supported.
next_time	timestamp	IN	No	Specifies the next time the job will be executed. The default value is the current system time (**sysdate**). If the specified time has past, the job is executed at the time it is submitted.
interval_time	text	IN	Yes	Calculates the next time to execute the job. It can be an interval expression, or **sysdate** followed by a numeric value, for example, **sysdate+1.0/24**. If this parameter is left empty or set to **null**, the job will be executed only once, and the job status will change to **'d'** afterward.
job	integer	OUT	No	Specifies the job ID. The value ranges from 1 to 32767. When **pkg_service.job_submit** is called using **select**, this parameter can be omitted.

- PKG_SERVICE.JOB_UPDATE

  The stored procedure UPDATE modifies user-definable attributes of a task, including the task content, next-execution time, and execution interval.

  The prototype of the PKG_SERVICE.JOB_UPDATE function is as follows:

```
PKG_SERVICE.JOB_UPDATE(
id            IN  BIGINT,
next_time     IN  TIMESTAMP,
interval_time IN  TEXT,
content       IN  TEXT);
```


Table 10-14 PKG_SERVICE.JOB_UPDATE parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	integer	IN	No	Specifies the job ID.
next_time	timestamp	IN	Yes	Specifies the next execution time. If this parameter is left empty, the system does not update the **next_time** parameter for the specified job. Otherwise, the system updates the **next_time** parameter for the specified job.
interval_time	text	IN	Yes	Specifies the time expression for calculating the next time the job will be executed. If this parameter is left empty, the system does not update the **interval_time** parameter for the specified job. Otherwise, the system updates the **interval_time** parameter for the specified job after necessary validity check. If this parameter is set to **null**, the job will be executed only once, and the job status will change to **'d'** afterward.
content	text	IN	Yes	Specifies the name of the stored procedure or SQL statement block that is executed. If this parameter is left empty, the system does not update the **content** parameter for the specified job. Otherwise, the system updates the **content** parameter for the specified job.

Example:

```
CREATE TABLE test_table(a int);
CREATE TABLE

CREATE OR REPLACE PROCEDURE test_job(a in int) IS
BEGIN
INSERT INTO test_table VALUES(a);
COMMIT;
END;
/
CREATE PROCEDURE

--PKG_SERVICE.JOB_SUBMIT
SELECT PKG_SERVICE.JOB_SUBMIT(NULL, 'call test_job(1);', to_date('20180101','yyyymmdd'),'sysdate
+1');
 job_submit
```



```
------------
     28269
(1 row)

SELECT PKG_SERVICE.JOB_SUBMIT(NULL, 'call test_job(1);', to_date('20180101','yyyymmdd'),'sysdate
+1.0/24');
 job_submit
------------
       1506
(1 row)

CALL PKG_SERVICE.JOB_SUBMIT(NULL, 'INSERT INTO test_table VALUES(1);  call test_job(1); call
test_job(1);', add_months(to_date('201701','yyyymm'),1), 'date_trunc(''day'',SYSDATE) + 1
+(8*60+30.0)/(24*60)' ,:jobid);
  job
-------
 14131
(1 row)

SELECT PKG_SERVICE.JOB_SUBMIT (101, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');
 job_submit
------------
       101
(1 row)

--PKG_SERVICE.JOB_UPDATE
CALL PKG_SERVICE.JOB_UPDATE(101, sysdate, 'sysdate + 1.0/1440', 'call test_job(1);');
 job_update
------------

(1 row)

CALL PKG_SERVICE.JOB_UPDATE(101, sysdate, 'sysdate + 1.0/1440', 'insert into test_table values(1);');
 job_update
------------

(1 row)

--PKG_SERVICE.JOB_FINISH
CALL PKG_SERVICE.JOB_FINISH(101,true);
 job_finish
------------

(1 row)

--PKG_SERVICE.JOB_CANCEL
CALL PKG_SERVICE.JOB_CANCEL(101);
 job_cancel
------------

(1 row)

DROP TABLE test_table;
DROP TABLE
```

- PKG_SERVICE.SUBMIT_ON_NODES

  Creates a scheduled job on a node. Only users sysadmin and monitor admin have this permission.

  The prototype of the PKG_SERVICE.SUBMIT_ON_NODES function is as follows:

```
PKG_SERVICE.SUBMIT_ON_NODES(
node_name    IN   NAME,
database     IN   NAME,
what         IN   TEXT,
next_date    IN   TIMESTAMP WITHOUT TIME ZONE,
job_interval IN   TEXT,
job          OUT  INTEGER);
```


Table 10-15 PKG_SERVICE.SUBMIT_ON_NODES parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
node_name	text	IN	No	Job execution node. Currently, the value can only be **'ALL_NODE'**, indicating that the job is executed on all nodes.
database	text	IN	No	Database used by a database instance job. When the node type is **'ALL_NODE'**, the value can only be **'postgres'**.
what	text	IN	No	Specifies the SQL statement to be executed. One or multiple DMLs, anonymous blocks, and statements for calling stored procedures, or all three combined are supported.
nextdate	timestamp	IN	No	Specifies the next time the job will be executed. The default value is the current system time (**sysdate**). If the specified time has past, the job is executed at the time it is submitted.
job_interval	text	IN	No	Calculates the next time to execute the job. It can be an interval expression, or **sysdate** followed by a numeric value, for example, **sysdate+1.0/24**. If this parameter is left empty or set to **NULL**, the job will be executed only once, and the job status will change to **'d'** afterward.
job	integer	OUT	No	Specifies the job ID. The value ranges from 1 to 32767. When dbms.submit_on_nodes is called using the SELECT statement, this parameter can be omitted.

Example:

```
SELECT pkg_service.submit_on_nodes('ALL_NODE', 'postgres', 'select
capture_view_to_json(''dbe_perf.statement'', 0);', sysdate, 'interval ''60 second''');
 submit_on_nodes
-----------------
           25376
(1 row)
```

- PKG_SERVICE.ISUBMIT_ON_NODES

  ISUBMIT_ON_NODES has the same syntax function as SUBMIT_ON_NODES, but the first parameter of ISUBMIT_ON_NODES is an input parameter, that is,

a specified task ID. In contrast, that last parameter of ISUBMIT_ON_NODES is an output parameter, indicating the task ID automatically generated by the system. Only users **sysadmin** and **monitor admin** have this permission.

- PKG_SERVICE.SQL_GET_ARRAY_RESULT

  This function is used to return the value of the bound OUT parameter of the array type and obtain the OUT parameter in a stored procedure.

  The prototype of the PKG_SERVICE.SQL_GET_ARRAY_RESULT function is as follows:

```
PKG_SERVICE.SQL_GET_ARRAY_RESULT(
   context_id in int,
   pos in VARCHAR2,
   column_value inout anyarray,
   result_type in anyelement
);
```

Table 10-16 PKG_SERVICE.SQL_GET_ARRAY_RESULT parameters

Parameter	Description
context_id	ID of the context to be queried
pos	Name of the bound parameter
column_value	Return value
result_type	Return type

- PKG_SERVICE.SQL_GET_VARIABLE_RESULT

  This function is used to return the value of the bound OUT parameter of the non-array type and obtain the OUT parameter in a stored procedure.

  The prototype of the PKG_SERVICE.SQL_GET_VARIABLE_RESULT function is as follows:

```
PKG_SERVICE.SQL_GET_VARIABLE_RESULT(
   context_id in int,
   pos in VARCHAR2,
   result_type in anyelement
)
RETURNS anyelement;
```

Table 10-17 PKG_SERVICE.SQL_GET_VARIABLE_RESULT parameters

Parameter	Description
context_id	ID of the context to be queried
pos	Name of the bound parameter
result_type	Return type

10.12.1.2 PKG_UTIL

Table 10-18 lists all APIs supported by PKG_UTIL.

Table 10-18 PKG_UTIL

API	Description
PKG_UTIL.LOB_GET_LENGTH	Obtains the length of a LOB.
PKG_UTIL.LOB_READ	Reads a part of a LOB.
PKG_UTIL.LOB_WRITE	Writes the source object to the target object in the specified format.
PKG_UTIL.LOB_APPEND	Appends the source LOB to the target LOB.
PKG_UTIL.LOB_COMPARE	Compares two LOBs based on the specified length.
PKG_UTIL.LOB_MATCH	Returns the position of the *M*th occurrence of a character string in a LOB.
PKG_UTIL.LOB_RESET	Resets the character in specified position of a LOB to a specified character.
PKG_UTIL.LOB_GET_LENGTH	Obtains and returns the specified length of a LOB.
PKG_UTIL.LOB_READ_HUGE	Reads a part of the LOB content based on the specified length and initial position offset, and returns the read LOB and length.
PKG_UTIL.LOB_WRITEAPPEND_HUGE	Reads the content of a specified length from the source BLOB or CLOB, appends the content to the target BLOB or CLOB, and returns the target object.
PKG_UTIL.LOB_APPEND_HUGE	Appends the source BLOB or CLOB to the target BLOB/CLOB and returns the target object.
PKG_UTIL.READ_BFILE_TO_BLOB	Loads the source BFILE file to the target BLOB and returns the target object.
PKG_UTIL.LOB_COPY_HUGE	Reads the content of a specified length from the specified offset position of the source BLOB or CLOB, writes the content to the specified offset position of the target BLOB or CLOB, and returns the target object.

API	Description
PKG_UTIL.BLOB_RESET	Sets a segment of data in a BLOB to the specified value and returns the processed BLOB and the actually processed length.
PKG_UTIL.CLOB_RESET	Sets a segment of data in a CLOB to spaces and returns the processed CLOB and the actually processed length.
PKG_UTIL.LOADBLOBFROMFILE	Reads the content of a specified length from the specified offset position of the source BFILE object, writes the content to the specified offset position of the target BLOB, and returns the target object, read position, and write position.
PKG_UTIL.LOADCLOBFROMFILE	Reads the content of a specified length from the specified offset position of the source BFILE object, writes the content to the specified offset position of the target CLOB, and returns the target object, read position, and write position.
PKG_UTIL.LOB_CONVERTTOBLOB_HUGE	Reads the content of a specified length from the specified offset position of the source CLOB, converts the content into a BLOB, and writes the BLOB to the specified position of target LOB. *amount* indicates the length to be converted.
PKG_UTIL.LOB_CONVERTTOCLOB_HUGE	Reads the content of a specified length from the specified offset position of the source CLOB, converts the content into a CLOB, and writes the CLOB to the specified position of target LOB. *amount* indicates the length to be converted.
PKG_UTIL.BFILE_GET_LENGTH	Obtains and returns the specified length of a BFILE file.
PKG_UTIL.BFILE_OPEN	Opens a BFILE file and returns its file descriptor.
PKG_UTIL.BFILE_CLOSE	Closes a BFILE file.

API	Description
PKG_UTIL.LOB_WRITE_HUGE	Reads the specified length of the source object from the start position, writes the content to the specified offset position of the target LOB, overrides the original content, and returns the target LOB.
PKG_UTIL.IO_PRINT	Displays character strings.
PKG_UTIL.RAW_GET_LENGTH	Obtains the length of RAW data.
PKG_UTIL.RAW_CAST_FROM_VARCHAR2	Converts VARCHAR2 data to RAW data.
•PKG_UTIL.RAW_CAST_FROM_...	Converts binary integers to RAW data.
•PKG_UTIL.RAW_CAST_TO_BI...	Converts RAW data to binary integers.
PKG_UTIL.RANDOM_SET_SEED	Sets a random seed.
PKG_UTIL.RANDOM_GET_VALUE	Returns a random value.
PKG_UTIL.FILE_SET_DIRNAME	Sets the directory to be operated.
PKG_UTIL.FILE_OPEN	Opens a file based on the specified file name and directory.
PKG_UTIL.FILE_SET_MAX_LINE_SIZE	Sets the maximum length of a line to be written to a file.
PKG_UTIL.FILE_IS_CLOSE	Checks whether a file handle is closed.
PKG_UTIL.FILE_READ	Reads data of a specified length from an open file handle.
PKG_UTIL.FILE_READLINE	Reads a line of data from an open file handle.
PKG_UTIL.FILE_WRITE	Writes the data specified in the buffer to a file.
PKG_UTIL.FILE_WRITELINE	Writes the buffer to a file and adds newline characters.
PKG_UTIL.FILE_NEWLINE	Adds a line.
PKG_UTIL.FILE_READ_RAW	Reads binary data of a specified length from an open file handle.
PKG_UTIL.FILE_WRITE_RAW	Writes binary data to a file.
PKG_UTIL.FILE_FLUSH	Writes data from a file handle to a physical file.
PKG_UTIL.FILE_CLOSE	Closes an open file handle.

API	Description
PKG_UTIL.FILE_REMOVE	Deletes a physical file. To do so, you must have the corresponding permission.
PKG_UTIL.FILE_RENAME	Renames a file on the disk, similar to **mv** in Unix.
PKG_UTIL.FILE_SIZE	Returns the size of a file.
PKG_UTIL.FILE_BLOCK_SIZE	Returns the number of blocks contained in a file.
PKG_UTIL.FILE_EXISTS	Checks whether a file exists.
PKG_UTIL.FILE_GETPOS	Specifies the offset of a returned file, in bytes.
PKG_UTIL.FILE_SEEK	Sets the offset for file position.
PKG_UTIL.FILE_CLOSE_ALL	Closes all file handles opened in a session.
PKG_UTIL.EXCEPTION_REPORT_ERROR	Throws an exception.
PKG_UTIL.RANDOM_SET_SEED	Sets a random seed.
PKG_UTIL.APP_READ_CLIENT_INFO	Reads the client information.
PKG_UTIL.APP_SET_CLIENT_INFO	Sets the client information.
PKG_UTIL.LOB_CONVERTTOBLOB	Converts the CLOB type to the BLOB type.
PKG_UTIL.LOB_CONVERTTOCLOB	Converts the BLOB type to the CLOB type.
PKG_UTIL.LOB_RAWTOTEXT	Converts the raw type to the TEXT type.
PKG_UTIL.LOB_RESET	Clears data of the LOB type.
PKG_UTIL.LOB_TEXTTORAW	Converts the TEXT type to the raw type.
PKG_UTIL.LOB_RAWTOTEXT	Converts the raw type to the TEXT type.
PKG_UTIL.LOB_WRITE	Writes data to the LOB type.
PKG_UTIL.MATCH_EDIT_DISTANCE_SIMILARITY	Calculates the difference between two character strings.
PKG_UTIL.RAW_CAST_TO_VARCHAR2	Converts the raw type to the VARCHAR2 type.

API	Description
PKG_UTIL.SESSION_CLEAR_CONTEXT	Clears the attribute values in **session_context**.
PKG_UTIL.SESSION_SEARCH_CONTEXT	Searches for an attribute value.
PKG_UTIL.SESSION_SET_CONTEXT	Sets an attribute value.
PKG_UTIL.UTILITY_FORMAT_CALL_STACK	Displays the call stack of a stored procedure.
PKG_UTIL.UTILITY_FORMAT_ERROR_BACKTRACE	Displays the error stack of a stored procedure.
PKG_UTIL.UTILITY_FORMAT_ERROR_STACK	Displays the error information about a stored procedure.
PKG_UTIL.UTILITY_GET_TIME	Displays the Unix timestamp of the system.
PKG_UTIL.UTILITY_COMPILE_SCHEMA	Recompiles specified schemas, packages, functions, and stored procedures. If an error is reported when compiling a PL/SQL object, the PL/SQL object is returned and the recompilation stops. This package has been discarded. **pkg_util.gs_compile_schema** is recommended.
PKG_UTIL.GS_COMPILE_SCHEMA	Recompiles specified schemas, packages, functions, and stored procedures. If an error is reported when compiling a PL/SQL object, the exception is captured and the recompilation continues to compile other objects until all objects are compiled or the number of recompilation attempts reaches the upper limit. When the advanced package is executed through JDBC, the SQLSTATE 00000 is displayed, which indicates that the operation is successful. For details about the error code description, see "Standard SQL Error Codes" in *Error Code Reference*.
PKG_UTIL.APP_SET_MODULE	Sets the value of a module.
PKG_UTIL.APP_READ_MODULE	Reads the value of a module.
PKG_UTIL.APP_SET_ACTION	Sets the value of an action.
PKG_UTIL.APP_READ_ACTION	Reads the value of an action.

API	Description
PKG_UTIL.MODIFY_PACKAGE_STATE	Modifies the PL/SQL status of the current session and releases all memory associated with each previously running PL/SQL program from the session.

- PKG_UTIL.LOB_GET_LENGTH

  Obtains the length of the input data.

  The prototype of the PKG_UTIL.LOB_GET_LENGTH function is as follows:

```
PKG_UTIL.LOB_GET_LENGTH(
lob       IN   CLOB
)
RETURN INTEGER;

PKG_UTIL.LOB_GET_LENGTH(
lob       IN   BLOB
)
RETURN INTEGER;
```

  Table 10-19 PKG_UTIL.LOB_GET_LENGTH parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
lob	CLOB/BLOB	IN	No	Indicates the object whose length is to be obtained.

- PKG_UTIL.LOB_READ

  Reads an object and returns the specified part.

  The prototype of the PKG_UTIL.LOB_READ function is as follows:

```
PKG_UTIL.LOB_READ(
lob       IN   ANYELEMENT,
len       IN   INT,
start     IN   INT,
mode      IN   INT
)
RETURN ANYELEMENT
```


Table 10-20 PKG_UTIL.LOB_READ parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
lob	CLOB / BLOB	IN	No	Specifies CLOB or BLOB data.
len	INT	IN	No	Specifies the length of the returned result.
start	INT	IN	No	Specifies the offset to the first character.
mode	INT	IN	No	Specifies the type of the read operation. **0** indicates **READ**, **1** indicates **TRIM**, and **2** indicates **SUBSTR**.

- PKG_UTIL.LOB_WRITE

  Writes the source object to the target object based on the specified parameters and returns the target object.

  The prototype of the PKG_UTIL.LOB_WRITE function is as follows:

```
PKG_UTIL.LOB_WRITE(
dest_lob    INOUT   BLOB,
src_lob     IN      RAW
len         IN      INT,
start_pos   IN      BIGINT
)
RETURN BLOB;
PKG_UTIL.LOB_WRITE(
dest_lob    INOUT   CLOB,
src_lob     IN      VARCHAR2
len         IN      INT,
start_pos       IN      BIGINT
)
RETURN CLOB;
```

Table 10-21 PKG_UTIL.LOB_WRITE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
dest_lob	CLOB / BLOB	INOUT	No	Specifies the target object that data will be written to.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
src_lob	CLOB/BLOB	IN	No	Specifies the source object to be written.
len	INT	IN	No	Specifies the write length of the source object.
start_pos	BIGINT	IN	No	Specifies the write start position of the target object.

- PKG_UTIL.LOB_APPEND

  Appends the source object to the target BLOB/CLOB and returns the target BLOB/CLOB.

  The prototype of the PKG_UTIL.LOB_APPEND function is as follows:

```
PKG_UTIL.LOB_APPEND(
dest_lob    INOUT   BLOB,
src_lob     IN      BLOB,
len         IN      INT DEFAULT NULL
)
RETURN BLOB;

PKG_UTIL.LOB_APPEND(
dest_lob    INOUT   CLOB,
src_lob     IN      CLOB,
len         IN      INT DEFAULT NULL
)
RETURN CLOB;
```

Table 10-22 PKG_UTIL.LOB_APPEND parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
dest_lob	BLOB/CLOB	INOUT	No	Specifies the target BLOB/CLOB that data will be written to.
src_lob	BLOB/CLOB	IN	No	Specifies the source BLOB/CLOB to be written.
len	INT	IN	Yes	Length read from *src* and appended to *dest*. The default value is null, indicating that all content of *src* is read and appended to *dest*.

- PKG_UTIL.LOB_COMPARE

  Checks whether objects are the same based on the specified start position and size. If **lob1** is larger, **1** is returned. If **lob2** is larger, **–1** is returned. If **lob1** is equal to **lob2**, **0** is returned.

  The prototype of the PKG_UTIL.LOB_COMPARE function is as follows:

```
PKG_UTIL.LOB_COMPARE(
lob1        IN   ANYELEMENT,
lob2        IN   ANYELEMENT,
len         IN   INT DEFAULT 1073741771,
start_pos1      IN   INT DEFAULT 1,
start_pos2      IN   INT DEFAULT 1
)
RETURN INTEGER;
```

  Table 10-23 PKG_UTIL.LOB_COMPARE parameters

Parameter	Type	Input / Output Parameter	Can Be Empty	Description
lob1	CLOB/ BLOB	IN	No	Indicates the character string for comparison.
lob2	CLOB/ BLOB	IN	No	Indicates the character string for comparison.
len	INT	IN	No	Indicates the length to be compared.
start_pos1	INT	IN	No	Specifies the start offset of **lob1**.
start_pos2	INT	IN	No	Specifies the start offset of **lob2**.

- PKG_UTIL.LOB_MATCH

  Returns the position where a pattern is displayed in a LOB for the *match_n*th time.

  The prototype of the PKG_UTIL.LOB_MATCH function is as follows:

```
PKG_UTIL.LOB_MATCH(
lob         IN   ANYELEMENT,
pattern      IN   ANYELEMENT,
start       IN   INT,
match_nth    IN   INT DEFAULT 1
)
RETURN INTEGER;
```


Table 10-24 PKG_UTIL.LOB_MATCH parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
lob	CLOB/BLOB	IN	No	Indicates the character string for comparison.
pattern	CLOB/BLOB	IN	No	Specifies the pattern to be matched.
start	INT	IN	No	Specifies the start position for LOB comparison.
match_nth	INT	IN	No	Specifies the matching times.

- PKG_UTIL.LOB_RESET

  Clears a character string and resets the string to the value of **value**.

  The prototype of the PKG_UTIL.LOB_RESET function is as follows:

```
PKG_UTIL.LOB_RESET(
lob         INOUT   BLOB,
len         INOUT   INT,
start       IN   INT DEFAULT 1,
value        IN   INT DEFAULT 0
)
RETURN RECORD;
```

Table 10-25 PKG_UTIL.LOB_RESET parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
lob	BLOB	IN	No	Indicates the character string for reset.
len	INT	IN	No	Specifies the length of the string to be reset.
start	INT	IN	No	Specifies the start position for reset.
value	INT	IN	Yes	Sets characters. Default value: **'0'**

- PKG_UTIL.LOB_GET_LENGTH

  Obtains and returns the specified length of a LOB.

  The prototype of the PKG_UTIL.LOB_GET_LENGTH function is as follows:

```
PKG_UTIL.LOB_GET_LENGTH(
   lob IN BLOB)
```



```
RETURN BIGINT;

PKG_UTIL.LOB_GET_LENGTH(
   lob IN CLOB)
RETURN BIGINT;
```

Table 10-26 PKG_UTIL.LOB_GET_LENGTH parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
lob	BLOB/CLOB	IN	No	LOB type

- PKG_UTIL.LOB_READ_HUGE

  Reads a part of the LOB content based on the specified length and initial position offset, and returns the read LOB and length.

  The prototype of the PKG_UTIL.LOB_READ_HUGE function is as follows:

```
PKG_UTIL.LOB_READ_HUGE(
   lob       IN CLOB,
   len       IN BIGINT,
   start_pos IN BIGINT,
   mode      IN INTEGER)
RETURN RECORD;

PKG_UTIL.LOB_READ_HUGE(
   lob       IN BLOB,
   len       IN BIGINT,
   start_pos IN BIGINT,
   mode      IN INTEGER)
RETURN RECORD;

PKG_UTIL.LOB_READ_HUGE(
   fd        IN INTEGER,
   len       IN BIGINT,
   start_pos IN BIGINT,
   mode      IN INTEGER)
RETURN RECORD;
```

Table 10-27 PKG_UTIL.LOB_READ_HUGE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
lob/fd	BLOB/CLOB/INTEGER	IN	No	File descriptor of the specified LOB or BFILE file.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
len	BIGINT	IN	No	Length to read.
start_pos	BIGINT	IN	No	Offset position from which read starts.
mode	INTEGER	IN	No	Read mode (**0**: read, **1**: trim, **2**: substr).

- PKG_UTIL.LOB_WRITEAPPEND_HUGE

  Reads the content of a specified length from the source BLOB or CLOB, appends the content to the target BLOB or CLOB, and returns the target object.

  The prototype of the PKG_UTIL.LOB_WRITEAPPEND_HUGE function is as follows:

```
PKG_UTIL.LOB_WRITEAPPEND_HUGE(
   dest_lob INOUT CLOB,
   len      IN    INTEGER,
   src_lob  IN    VARCHAR2
)RETURN CLOB;

PKG_UTIL.LOB_WRITEAPPEND_HUGE(
   dest_lob INOUT BLOB,
   len      IN    INTEGER,
   src_lob  IN    RAW
)RETURN BLOB;
```

  Table 10-28 PKG_UTIL.LOB_WRITEAPPEND_HUGE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
dest_lob	BLOB/CLOB	INOUT	No	Target BLOB/CLOB to which data is written.
len	INTEGER	IN	Yes	Length of the source object to be written. If the value is **NULL**, the entire source object is written by default.
src_lob	VARCHAR2/RAW	IN	No	BLOB/CLOB from which data is to be written.

- PKG_UTIL.LOB_APPEND_HUGE

  Appends the source BLOB or CLOB to the target BLOB/CLOB and returns the target object.

  The prototype of the PKG_UTIL.LOB_APPEND_HUGE function is as follows:

```
PKG_UTIL.LOB_APPEND_HUGE(
   dest_lob INOUT BLOB,
   src_lob  IN    BLOB)
RETURN BLOB;
PKG_UTIL.LOB_APPEND_HUGE(
   dest_lob INOUT CLOB,
   src_lob  IN    CLOB)
RETURN CLOB;
```

  Table 10-29 PKG_UTIL.LOB_APPEND_HUGE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
dest_lob	BLOB/CLOB	INOUT	No	Target BLOB/CLOB to which data is written.
src_lob	BLOB/CLOB	IN	No	BLOB/CLOB from which data is to be written.

- PKG_UTIL.READ_BFILE_TO_BLOB

  Loads the source BFILE file to the target BLOB and returns the target object.

  The prototype of the PKG_UTIL.READ_BFILE_TO_BLOB function is as follows:

```
PKG_UTIL.READ_BFILE_TO_BLOB(
   fd IN INTEGER
)RETURN BLOB;
```

  Table 10-30 PKG_UTIL.READ_BFILE_TO_BLOB parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
fd	INTEGER	IN	No	Source BFILE file to be read.

- PKG_UTIL.LOB_COPY_HUGE

  Reads the content of a specified length from the specified offset position of the source BLOB or CLOB, writes the content to the specified offset position of the target BLOB or CLOB, and returns the target object.

  The prototype of the PKG_UTIL.LOB_COPY_HUGE function is as follows:

```
PKG_UTIL.LOB_COPY_HUGE(
   lob_obj    INOUT BLOB,
   source_obj  IN    BLOB,
   amount      IN    BIGINT,
   dest_offset IN    BIGINT DEFAULT 1,
   src_offset  IN    BIGINT DEFAULT 1
)RETURN BLOB;

PKG_UTIL.LOB_COPY_HUGE(
   lob_obj    INOUT CLOB,
   source_obj  IN    CLOB,
   amount      IN    BIGINT,
   dest_offset IN    BIGINT DEFAULT 1,
   src_offset  IN    BIGINT DEFAULT 1
)RETURN CLOB;
```

Table 10-31 PKG_UTIL.LOB_COPY_HUGE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
lob_obj	BLOB/CLOB	INOUT	No	Target BLOB/CLOB.
source_obj	BLOB/CLOB	IN	No	Source BLOB/CLOB.
amount	BIGINT	IN	No	Length of the data to be copied (in bytes for BLOBs or in characters for CLOBs).
dest_offset	BIGINT	IN	No	Offset position of the target LOB to which the data is loaded.
src_offset	BIGINT	IN	No	Offset position of the source LOB from which the data is read.

- PKG_UTIL.BLOB_RESET

  Sets a segment of data in a BLOB to the specified value and returns the processed BLOB and the actually processed length.

  The prototype of the PKG_UTIL.BLOB_RESET function is as follows:

```
PKG_UTIL.BLOB_RESET(
   lob       INOUT BLOB,
   len       INOUT BIGINT,
   start_pos IN    BIGINT DEFAULT 1,
   value     IN    INTEGER DEFAULT 0
)RETURN RECORD;
```


Table 10-32 PKG_UTIL.BLOB_RESET parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
lob	BLOB	INOUT	No	LOB to be reset.
len	INTEGER	INOUT	No	Length to reset, in bytes.
start	INTEGER	IN	No	Specifies the start position for reset.
value	INTEGER	IN	Yes	Sets characters. Default value: **'0'**.

- PKG_UTIL.CLOB_RESET

  Sets a piece of data to spaces.

  The prototype of the PKG_UTIL.CLOB_RESET function is as follows:

```
PKG_UTIL.CLOB_RESET(
   lob       INOUT CLOB,
   len       INOUT BIGINT,
   start_pos IN    BIGINT DEFAULT 1
)RETURN RECORD;
```

Table 10-33 PKG_UTIL.CLOB_RESET parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
lob	CLOB	INOUT	No	LOB to be reset.
len	INTEGER	INOUT	No	Length to reset, in characters.
start	INTEGER	IN	No	Reset start position. The default value is **1**.

- PKG_UTIL.LOADBLOBFROMFILE

  Reads the content of a specified length from the specified offset position of the source BFILE object, writes the content to the specified offset position of the target BLOB, and returns the target object, read position, and write position.

The prototype of the PKG_UTIL.LOADBLOBFROMFILE function is as follows:

```
PKG_UTIL.LOADBLOBFROMFILE(
   dest_lob     INOUT  BLOB,
   fd           IN     INTEGER,
   amount       IN     BIGINT,
   dest_offset  INOUT  BIGINT,
   file_offset  INOUT  BIGINT
)RETURN RECORD;
```

Table 10-34 PKG_UTIL.LOADBLOBFROMFILE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
dest_lob	BLOB	INOUT	No	Target BLOB as the INOUT parameter.
fd	INTEGER	IN	No	File descriptor of the source BFILE object.
amount	BIGINT	IN	No	Length of the data to be copied (in bytes for BLOBs or in characters for CLOBs).
dest_offset	BIGINT	INOUT	No	Offset position of the target LOB to which the data is written as the INOUT parameter.
src_offset	BIGINT	INOUT	No	Offset position of the source BFILE file from which the data is read as the INOUT parameter.

- PKG_UTIL.LOADCLOBFROMFILE

  Reads the content of a specified length from the specified offset position of the source BFILE object, writes the content to the specified offset position of the target CLOB, and returns the target object, read position, and write position.

  The prototype of the PKG_UTIL.LOADCLOBFROMFILE function is as follows:

```
PKG_UTIL.LOADCLOBFROMFILE(
   dest_lob     INOUT  CLOB,
   fd           IN     INTEGER,
   amount       IN     BIGINT,
   dest_offset  INOUT  BIGINT,
   file_offset  INOUT  BIGINT
)RETURN RECORD;
```


Table 10-35 PKG_UTIL.LOADCLOBFROMFILE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
dest_lob	CLOB	INOUT	No	Target CLOB as the INOUT parameter.
fd	INTEGER	IN	No	File descriptor of the source BFILE object.
amount	BIGINT	IN	No	Length to copy (in characters for CLOBs).
dest_offset	BIGINT	INOUT	No	Offset position of the target LOB to which the data is written as the INOUT parameter.
src_offset	BIGINT	INOUT	No	Offset position of the source BFILE file from which the data is read as the INOUT parameter.

- PKG_UTIL.LOB_CONVERTTOBLOB_HUGE

  Reads the content of a specified length from the specified offset position of the source CLOB, converts the content into a BLOB, and writes the BLOB to the specified position of target LOB. *amount* indicates the length to be converted.

  The prototype of the PKG_UTIL.LOB_CONVERTTOBLOB_HUGE function is as follows:

```
PKG_UTIL.LOB_CONVERTTOBLOB_HUGE(
   dest_lob    INOUT  BLOB,
   src_clob    IN     CLOB,
   amount      IN     BIGINT,
   dest_offset INOUT  BIGINT,
   src_offset  INOUT  BIGINT)
)RETURN RECORD;
```


Table 10-36 PKG_UTIL.LOB_CONVERTTOBLOB_HUGE parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
dest_lob	BLOB	INOUT	No	Target LOB.
src_clob	CLOB	IN	No	CLOB to be converted.
amount	BIGINT	IN	No	Length to convert, in characters.
dest_offset	BIGINT	INOUT	No	Offset position of the target LOB to which the data is written as the INOUT parameter.
src_offset	BIGINT	INOUT	No	Offset position of the source CLOB from which the data is read as the INOUT parameter.

- PKG_UTIL.LOB_CONVERTTOCLOB_HUGE

  Reads the content of a specified length from the specified offset position of the source CLOB, converts the content into a CLOB, and writes the CLOB to the specified position of target LOB. *amount* indicates the length to be converted.

  The prototype of the PKG_UTIL.LOB_CONVERTTOCLOB_HUGE function is as follows:

```
PKG_UTIL.LOB_CONVERTTOCLOB_HUGE(
   dest_lob    INOUT  CLOB,
   src_blob    IN     BLOB,
   amount      IN     BIGINT,
   dest_offset INOUT  BIGINT,
   src_offset  INOUT  BIGINT)
)RETURN RECORD;
```


Table 10-37 PKG_UTIL.LOB_CONVERTTOCLOB_HUGE parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
dest_lob	CLOB	INOUT	No	Target LOB.
src_blob	BLOB	IN	No	BLOB to be converted.
amount	BIGINT	IN	No	Length to convert, in bytes.
dest_offset	BIGINT	INOUT	No	Offset position of the target LOB to which the data is written as the INOUT parameter.
src_offset	BIGINT	INOUT	No	Offset position of the source CLOB from which the data is read as the INOUT parameter.

- PKG_UTIL.BFILE_GET_LENGTH

  Obtains and returns the specified length of a BFILE file.

  The prototype of the PKG_UTIL.BFILE_GET_LENGTH function is as follows:

```
PKG_UTIL.BFILE_GET_LENGTH(
    fd INTEGER
)RETURN BIGINT;
```


Table 10-38 PKG_UTIL.LOB_GET_LENGTH parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
fd	INTEGER	IN	No	File descriptor of the specified BFILE file.

- PKG_UTIL.BFILE_OPEN

  Opens a BFILE file and returns its file descriptor.

  The prototype of the PKG_UTIL.BFILE_OPEN function is as follows:

```
PKG_UTIL.BFILE_OPEN(
    file_name TEXT,
    open_mode TEXT)
RETURN INTEGER;
```

Table 10-39 PKG_UTIL.BFILE_OPEN parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file_name	TEXT	IN	No	Name of the specified BFILE file.
open_mode	TEXT	IN	No	Open mode, which can only be set to **r**, that is, the read mode.

- PKG_UTIL.BFILE_CLOSE

  Closes a BFILE file.

  The prototype of the PKG_UTIL.BFILE_CLOSE function is as follows:

```
PKG_UTIL.BFILE_CLOSE(
    fd INTEGER)
RETURN bool;
```

Table 10-40 PKG_UTIL.BFILE_CLOSE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
fd	INTEGER	IN	No	File descriptor of the specified BFILE file.

- PKG_UTIL.LOB_WRITE_HUGE

Reads the specified length of the source object from the start position, writes the content to the specified offset position of the target LOB, overrides the original content, and returns the target LOB.

The prototype of the PKG_UTIL.LOB_WRITE_HUGE function is as follows:

```
PKG_UTIL.LOB_WRITE_HUGE(
   dest_lob  INOUT BLOB,
   len       IN INTEGER,
   start_pos IN BIGINT,
   src_lob   IN RAW
)RETURN BLOB;

PKG_UTIL.LOB_WRITE_HUGE(
   dest_lob  INOUT CLOB,
   len       IN INTEGER,
   start_pos IN BIGINT,
   src_lob   IN VARCHAR2
)RETURN CLOB;
```

Table 10-41 PKG_UTIL.LOB_WRITE_HUGE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
dest_lob	BLOB/CLOB	INOUT	No	Target LOB as the INOUT parameter, to which the content is to be written.
len	INTEGER	IN	No	Length of the data to be written (in bytes for BLOBs or in characters for CLOBs)
start_pos	BIGINT	IN	No	Offset position for writing data to *dest_lob*
src_lob	RAW/VARCHAR2	IN	No	Source LOB.

- PKG_UTIL.IO_PRINT

  Outputs a string.

  The prototype of the PKG_UTIL.IO_PRINT function is as follows:

  ```
  PKG_UTIL.IO_PRINT(
  format       IN   TEXT,
  is_one_line  IN   BOOLEAN
  )
  RETURN void;
  ```


Table 10-42 PKG_UTIL.IO_PRINT parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
format	TEXT	IN	No	Indicates the character string to be output.
is_one_line	BOOLEAN	IN	No	Specifies whether to output the string as a line.

- PKG_UTIL.RAW_GET_LENGTH

  Obtains the length of RAW data.

  The prototype of the PKG_UTIL.RAW_GET_LENGTH function is as follows:

```
PKG_UTIL.RAW_GET_LENGTH(
value       IN   RAW
)
RETURN INTEGER;
```

Table 10-43 PKG_UTIL.RAW_GET_LENGTH parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
raw	RAW	IN	No	Indicates the object whose length is to be obtained.

- PKG_UTIL.RAW_CAST_FROM_VARCHAR2

  Converts VARCHAR2 data to RAW data.

  The prototype of the PKG_UTIL.RAW_CAST_FROM_VARCHAR2 function is as follows:

```
PKG_UTIL.RAW_CAST_FROM_VARCHAR2(
str       IN   VARCHAR2
)
RETURN raw;
```

Table 10-44 PKG_UTIL.RAW_CAST_FROM_VARCHAR2 parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
str	VARCHAR2	IN	No	Indicates the source data to be converted

- PKG_UTIL.RAW_CAST_FROM_BINARY_INTEGER

  Converts BIGINT data to RAW data.

  The prototype of the PKG_UTIL.RAW_CAST_FROM_BINARY_INTEGER function is as follows:

```
PKG_UTIL.RAW_CAST_FROM_BINARY_INTEGER(
value       IN   BIGINT,
endianess   IN   INTEGER
)
RETURN RAW;
```

  Table 10-45 PKG_UTIL.RAW_CAST_FROM_BINARY_INTEGER parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
value	BIGINT	IN	No	Indicates the source data to be converted
endianess	INTEGER	IN	No	The value is an integer in lexicographic order. Currently, the value can be **1** (BIG_ENDIAN), **2** (LITTLE_ENDIAN), or **3** (MACHINE_ENDIAN).

- PKG_UTIL.RAW_CAST_TO_BINARY_INTEGER

  Converts RAW data into BINARY_INTEGER.

  The prototype of the PKG_UTIL.RAW_CAST_TO_BINARY_INTEGER function is as follows:

```
PKG_UTIL.RAW_CAST_TO_BINARY_INTEGER(
value       IN   RAW,
endianess   IN   INTEGER
)
RETURN INTEGER;
```

  Table 10-46 PKG_UTIL.RAW_CAST_TO_BINARY_INTEGER parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
value	RAW	IN	No	Indicates the source data to be converted
endianess	INTEGER	IN	No	The value is an integer in lexicographic order. Currently, the value can be **1** (BIG_ENDIAN), **2** (LITTLE_ENDIAN), or **3** (MACHINE_ENDIAN).

- PKG_UTIL.RANDOM_SET_SEED

  Sets a random seed.

  The prototype of the PKG_UTIL.RANDOM_SET_SEED function is as follows:


```
PKG_UTIL.RANDOM_SET_SEED(
seed        IN   INT
)
RETURN integer;
```

Table 10-47 PKG_UTIL.RANDOM_SET_SEED parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
seed	INT	IN	No	Sets a random seed.

- PKG_UTIL.RANDOM_GET_VALUE

  Returns a 15-digit random number ranging from 0 to 1.

  The prototype of the PKG_UTIL.RANDOM_GET_VALUE function is as follows:

```
PKG_UTIL.RANDOM_GET_VALUE(
)
RETURN NUMERIC;
```

- PKG_UTIL.FILE_SET_DIRNAME

  Sets the directory to be operated. It must be called to set directory for each operation involving a single directory.

  The prototype of the PKG_UTIL.FILE_SET_DIRNAME function is as follows:

```
PKG_UTIL.FILE_SET_DIRNAME(
dir  IN  TEXT
)
RETURN BOOL
```

Table 10-48 PKG_UTIL.FILE_SET_DIRNAME parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
dirname	TEXT	IN	No	Directory of a file. It is a string, indicating an object name. **NOTE** File directories need to be added to the system catalog **PG_DIRECTORY**. If the input path does not match the path in **PG_DIRECTORY**, an error indicating that the path does not exist is reported. Functions that involve **location** as parameters also comply with this rule.

- PKG_UTIL.FILE_OPEN

  Opens a file. A maximum of 50 files can be opened at a time. This function returns a handle of the INTEGER type.

  The prototype of the PKG_UTIL.FILE_OPEN function is as follows:

  ```
  PKG_UTIL.FILE_OPEN(
  file_name    IN  TEXT,
  open_mode    IN  INTEGER)
  ```

  Table 10-49 PKG_UTIL.FILE_OPEN parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
file_name	TEXT	IN	No	File name with an extension (file type), excluding the path name. A path contained in a file name is ignored in the OPEN function. In Unix, the file name cannot end with the combination of a slash and a dot (/.).
open_mode	INTEGER	IN	No	File opening mode, including **r** (read), **w** (write), and **a** (append). **NOTE** For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits.

- PKG_UTIL.FILE_SET_MAX_LINE_SIZE

  Sets the maximum length of a line to be written to a file.

  The prototype of the PKG_UTIL.FILE_SET_MAX_LINE_SIZE function is as follows:

  ```
  PKG_UTIL.FILE_SET_MAX_LINE_SIZE(
  max_line_size in INTEGER)
  RETURN BOOL
  ```


Table 10-50 PKG_UTIL.FILE_SET_MAX_LINE_SIZE parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
max_line_size	INTEGER	IN	No	Maximum number of characters in each line, including newline characters. The minimum value is **1** and the maximum is **32767**. If this parameter is not specified, the default value **1024** is used.

- PKG_UTIL.FILE_IS_CLOSE

  Checks whether a file handle is closed.

  The prototype of the PKG_UTIL.FILE_IS_CLOSE function is as follows:

```
PKG_UTIL.FILE_IS_CLOSE(
file in INTEGER
)
RETURN BOOL
```

Table 10-51 PKG_UTIL.FILE_IS_CLOSE parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
file	INTEGER	IN	No	Opened file handle

- PKG_UTIL.FILE_READ

  Reads a line of data from an open file handle based on the specified length.

  The prototype of the PKG_UTIL.FILE_READ function is as follows:

```
PKG_UTIL.FILE_READ(
file     IN   INTEGER,
```



```
buffer   OUT  TEXT,
len      IN   BIGINT DEFAULT 1024)
```

Table 10-52 PKG_UTIL.FILE_READ parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
file	INTEGER	IN	No	File handle opened by calling the OPEN function. The file must be opened in read mode. Otherwise, the **INVALID_OPERATION** exception is thrown.
buffer	TEXT	IN	No	Buffer used to receive data
len	BIGINT	IN	Yes	Number of bytes read from a file

- PKG_UTIL.FILE_READLINE

  Reads a line of data from an open file handle based on the specified length.

  The prototype of the PKG_UTIL.FILE_READLINE function is as follows:

  ```
  PKG_UTIL.FILE_READLINE(
  file    IN  INTEGER,
  buffer  OUT TEXT,
  len     IN  INTEGER DEFAULT NULL)
  ```


Table 10-53 PKG_UTIL.FILE_READLINE parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
file	INTEGER	IN	No	File handle opened by calling the OPEN function. The file must be opened in read mode. Otherwise, the **INVALID_OPERATION** exception is thrown.
buffer	TEXT	IN	No	Buffer used to receive data
len	INTEGER	IN	Yes	Number of bytes read from a file. The default value is **NULL**. If the default value **NULL** is used, **max_line_size** is used to specify the line size.

- PKG_UTIL.FILE_WRITE

  Writes the data specified in the buffer to a file.

  The prototype of the PKG_UTIL.FILE_WRITE function is as follows:

  ```
  PKG_UTIL.FILE_WRITE(
  file IN INTEGER,
  buffer IN TEXT
  )
  RETURN BOOL
  ```


Table 10-54 PKG_UTIL.FILE_WRITE parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
file	INTEGER	IN	No	Opened file handle
buffer	TEXT	IN	No	Text data to be written to a file. The maximum buffer size is 32767 bytes. If no value is specified, the default value is 1024 bytes. Before the writing is performed, the buffer occupied by PUT operations cannot exceed 32767 bytes. **NOTE** For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits.

- PKG_UTIL.FILE_NEWLINE

  Writes a line terminator to an open file. The line terminator is related to the platform.

  The prototype of the PKG_UTIL.FILE_NEWLINE function is as follows:

```
PKG_UTIL.FILE_NEWLINE(
file IN INTEGER
)
RETURN BOOL
```


Table 10-55 PKG_UTIL.FILE_NEWLINE parameters

Parameter	Type	Input/OutputParameter	CanBeEmpty	Description
file	INTEGER	IN	No	Opened file handle

- PKG_UTIL.FILE_WRITELINE

  Writes a line to a file.

  The prototype of the PKG_UTIL.FILE_WRITELINE function is as follows:

```
PKG_UTIL.FILE_WRITELINE(
file IN INTEGER,
buffer IN TEXT
)
RETURN BOOL
```

Table 10-56 PKG_UTIL.FILE_WRITELINE parameters

Parameter	Type	Input/OutputParameter	CanBeEmpty	Description
file	INTEGER	IN	No	Opened file handle
buffer	TEXT	IN	No	Content to be written.

- PKG_UTIL.FILE_READ_RAW

  Reads binary data of a specified length from an open file handle and returns the read binary data. The return type is RAW.

The prototype of the PKG_UTIL.FILE_READ_RAW function is as follows:

```
PKG_UTIL.FILE_READ_RAW(
file      IN INTEGER,
length    IN INTEGER DEFAULT NULL
)
RETURN raw
```

Table 10-57 PKG_UTIL.FILE_READ_RAW parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
file	INTEGER	IN	No	Opened file handle
length	INTEGER	IN	Yes	Length of the data to be read. The default value is **NULL**. By default, all data in the file is read. The maximum size is 1 GB.

- PKG_UTIL.FILE_WRITE_RAW

  Writes the input binary object of the RAW type to an open file. If the insertion is successful, **true** is returned.

  The prototype of the PKG_UTIL.FILE_WRITE_RAW function is as follows:

  ```
  PKG_UTIL.FILE_WRITE_RAW(
  file IN INTEGER,
  r    IN RAW
  )
  RETURN BOOL
  ```


Table 10-58 PKG_UTIL.FILE_NEWLINE parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
file	INTEGER	IN	No	Opened file handle
r	RAW	IN	Yes	Data to be written to the file **NOTE** For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits.

- PKG_UTIL.FILE_FLUSH

  Data in a file handle must be written into a physical file. Data in the buffer must have a line terminator. Refresh is important if a file must be read when it is opened. For example, debugging information can be refreshed to a file so that it can be read immediately.

  The prototype of the PKG_UTIL.FILE_FLUSH function is as follows:

  ```
  PKG_UTIL.FILE_FLUSH (
  file IN INTEGER
  )
  RETURN VOID
  ```


Table 10-59 PKG_UTIL.FILE_FLUSH parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
file	INTEGER	IN	No	Opened file handle

- PKG_UTIL.FILE_CLOSE

  Closes an open file handle.

  The prototype of the PKG_UTIL.FILE_CLOSE function is as follows:

  ```
  PKG_UTIL.FILE_CLOSE (
  file IN INTEGER
  )
  RETURN BOOL
  ```

  Table 10-60 PKG_UTIL.FILE_CLOSE parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
file	INTEGER	IN	No	Opened file handle

- PKG_UTIL.FILE_REMOVE

  Deletes a disk file. To perform this operation, you must have required permissions.

  The prototype of the PKG_UTIL.FILE_REMOVE function is as follows:

  ```
  PKG_UTIL.FILE_REMOVE(
  file_name IN TEXT
  ```



```
)
RETURN VOID
```

Table 10-61 PKG_UTIL.FILE_REMOVE parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
filen_ame	TEXT	IN	No	Name of the file to be deleted

- PKG_UTIL.FILE_RENAME

  Renames a file on the disk, similar to **mv** in Unix.

  The prototype of the PKG_UTIL.FILE_RENAME function is as follows:

  ```
  PKG_UTIL.FILE_RENAME(
  src_dir IN TEXT,
  src_file_name IN TEXT,
  dest_dir IN TEXT,
  dest_file_name IN TEXT,
  overwrite BOOLEAN DEFAULT FALSE)
  ```


Table 10-62 PKG_UTIL.FILE_RENAME parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
src_dir	TEXT	IN	Yes	Source file directory (case-sensitive) **NOTE** • Location of the file directory, which needs to be added to system catalog **PG_DIRECTORY**. If the input path does not match the path in **PG_DIRECTORY**, an error indicating that the path does not exist is reported. • When the GUC parameter **safe_data_path** is enabled, you can only use an advanced package to operate files in the file path specified by **safe_data_path**.
src_file_name	TEXT	IN	Yes	Source file name
dest_dir	TEXT	IN	Yes	Target file directory (case-sensitive) **NOTE** • Location of the file directory, which needs to be added to system catalog **PG_DIRECTORY**. If the input path does not match the path in **PG_DIRECTORY**, an error indicating that the path does not exist is reported. • When the GUC parameter **safe_data_path** is enabled, you can only use an advanced package to operate files in the file path specified by **safe_data_path**.
dest_file_name	TEXT	IN	Yes	Target file name
overwrite	BOOLEAN	IN	No	The default value is **false**. If a file with the same name exists in the destination directory, the file will not be rewritten.

- PKG_UTIL.FILE_SIZE

  Returns the size of a specified file.

  The prototype of the PKG_UTIL.FILE_SIZE function is as follows:


```
BIGINT PKG_UTIL.FILE_SIZE(
file_name IN TEXT
)RETURN BIGINT
```

Table 10-63 PKG_UTIL.FILE_SIZE parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
file_name	TEXT	IN	No	File name

- PKG_UTIL.FILE_BLOCK_SIZE

  Returns the number of blocks contained in a specified file.

  The prototype of the PKG_UTIL.FILE_BLOCK_SIZE function is as follows:

  ```
  BIGINT PKG_UTIL.FILE_BLOCK_SIZE(
  file_name IN TEXT
  )RETURN BIGINT
  ```

  Table 10-64 PKG_UTIL.FILE_BLOCK_SIZE parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
file_name	TEXT	IN	No	File name

- PKG_UTIL.FILE_EXISTS

  Checks whether a file exists.

  The prototype of the PKG_UTIL.FILE_EXISTS function is as follows:

  ```
  PKG_UTIL.FILE_EXISTS(
  file_name IN TEXT
  )
  RETURN BOOL
  ```


Table 10-65 PKG_UTIL.FILE_EXISTS parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
file_name	TEXT	IN	No	File name

- PKG_UTIL.FILE_GETPOS

  Specifies the offset of a returned file, in bytes.

  The prototype of the PKG_UTIL.FILE_GETPOS function is as follows:

```
PKG_UTIL.FILE_GETPOS(
file IN INTEGER
)
RETURN BIGINT
```

Table 10-66 PKG_UTIL.FILE_GETPOS parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
file	INTEGER	IN	No	Opened file handle

- PKG_UTIL.FILE_SEEK

  Adjusts the position of a file pointer forward or backward based on the specified number of bytes.

  The prototype of the PKG_UTIL.FILE_SEEK function is as follows:

```
VOID PKG_UTIL.FILE_SEEK(
file IN INTEGER,
start IN BIGINT
)
RETURN VOID
```


Table 10-67 PKG_UTIL.FILE_SEEK parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
file	INTEGER	IN	No	Opened file handle
start	BIGINT	IN	No	File offset, in bytes.

- PKG_UTIL.FILE_CLOSE_ALL

  Closes all file handles opened in a session.

  The prototype of the PKG_UTIL.FILE_CLOSE_ALL function is as follows:

  ```
  PKG_UTIL.FILE_CLOSE_ALL(
  )
  RETURN VOID
  ```

  Table 10-68 PKG_UTIL.FILE_CLOSE_ALL parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
None	None	None	None	None

- PKG_UTIL.EXCEPTION_REPORT_ERROR

  Throws an exception.

The prototype of the PKG_UTIL.EXCEPTION_REPORT_ERROR function is as follows:

```
PKG_UTIL.EXCEPTION_REPORT_ERROR(
code INTEGER,
log TEXT,
flag BOOLEAN DEFAULT FALSE
)
RETURN INTEGER
```

Table 10-69 PKG_UTIL.EXCEPTION_REPORT_ERROR parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
code	INTEGER	IN	No	Error code displayed when a running exception occurs.
log	TEXT	IN	No	Log information displayed when a running exception occurs.
flag	BOOLEAN	IN	Yes	Reserved. The default value is **false**.

- PKG_UTIL.APP_READ_CLIENT_INFO

  Reads the client information.

  The prototype of the PKG_UTIL.app_read_client_info function is as follows:

  ```
  PKG_UTIL.app_read_client_info(
  OUT buffer TEXT
  )RETURN TEXT
  ```


Table 10-70 PKG_UTIL.APP_READ_CLIENT_INFO parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
buffer	TEXT	IN	No	Client information returned.

- PKG_UTIL.APP_SET_CLIENT_INFO

  Sets the client information.

  The prototype of the PKG_UTIL.APP_SET_CLIENT_INFO function is as follows:

```
PKG_UTIL.APP_SET_CLIENT_INFO(
str TEXT
)
```

Table 10-71 PKG_UTIL.APP_SET_CLIENT_INFO parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
str	TEXT	IN	No	Client information to be set.

- PKG_UTIL.LOB_CONVERTTOBLOB

  Converts a CLOB to a BLOB. **amount** indicates the conversion length.

  The prototype of the PKG_UTIL.LOB_CONVERTTOBLOB function is as follows:

```
PKG_UTIL.lob_converttoblob(
dest_lob BLOB,
src_clob CLOB,
amount INTEGER,
dest_offset INTEGER,
src_offset INTEGER
)RETURN RAW
```


Table 10-72 PKG_UTIL.LOB_CONVERTTOBLOB parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
dest_lob	BLOB	IN	No	Target LOB.
src_clob	CLOB	IN	No	CLOB to be converted.
amount	INTEGER	IN	No	Conversion length.
dest_offset	INTEGER	IN	No	Start position of the target LOB.
src_offset	INTEGER	IN	No	Start position of the source CLOB.

- PKG_UTIL.LOB_CONVERTTOCLOB

  Converts a BLOB to a CLOB. **amount** indicates the conversion length.

  The prototype of the PKG_UTIL.lob_converttoclob function is as follows:

```
PKG_UTIL.lob_converttoclob(
dest_lob CLOB,
src_blob BLOB,
amount INTEGER,
dest_offset INTEGER,
src_offset INTEGER
)RETURN TEXT
```


Table 10-73 PKG_UTIL.LOB_CONVERTTOCLOB parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
dest_lob	CLOB	IN	No	Target LOB.
src_blob	BLOB	IN	No	CLOB to be converted.
amount	INTEGER	IN	No	Conversion length.
dest_offset	INTEGER	IN	No	Start position of the target LOB.
src_offset	INTEGER	IN	No	Start position of the source CLOB.

- PKG_UTIL.LOB_TEXTTORAW

  Converts TEXT data into RAW data.

  The prototype of the PKG_UTIL.LOB_TEXTTORAW function is as follows:

  ```
  PKG_UTIL.LOB_TEXTTORAW(
  src_lob CLOB
  )
  RETURN RAW
  ```


Table 10-74 PKG_UTIL.LOB_TEXTTORAW parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
src_lob	CLOB	IN	No	LOB to be converted.

- PKG_UTIL.LOB_RAWTOTEXT

  Converts RAW data to TEXT data.

  The prototype of the PKG_UTIL.LOB_RAWTOTEXT function is as follows:

```
PKG_UTIL.LOB_RAWTOTEXT(
src_lob IN BLOB
)
RETURN TEXT
```

Table 10-75 PKG_UTIL.LOB_RAWTOTEXT parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
src_lob	BLOB	IN	No	LOB to be converted.

- PKG_UTIL.MATCH_EDIT_DISTANCE_SIMILARITY

  Calculates the difference between two character strings.

  The prototype of the PKG_UTIL.MATCH_EDIT_DISTANCE_SIMILARITY function is as follows:

```
PKG_UTIL.MATCH_EDIT_DISTANCE_SIMILARITY(
str1 TEXT,
str2 TEXT
)
RETURN INTEGER
```


Table 10-76 PKG_UTIL.MATCH_EDIT_DISTANCE_SIMILARITY parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
str1	TEXT	IN	No	First character string.
str2	TEXT	IN	No	Second character string.

- PKG_UTIL.RAW_CAST_TO_VARCHAR2

  Converts RAW data into VARCHAR2 data.

  The prototype of the PKG_UTIL.RAW_CAST_TO_VARCHAR2 function is as follows:

  ```
  PKG_UTIL.RAW_CAST_TO_VARCHAR2(
  str RAW
  )
  RETURN VARCHAR2
  ```

  Table 10-77 PKG_UTIL.RAW_CAST_TO_VARCHAR2 parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
str	RAW	IN	No	Hexadecimal string

- PKG_UTIL.SESSION_CLEAR_CONTEXT

  Clears the session context.

  The prototype of the PKG_UTIL.SESSION_CLEAR_CONTEXT function is as follows:

  ```
  PKG_UTIL.SESSION_CLEAR_CONTEXT(
  namespace TEXT,
  ```



```
client_identifier TEXT,
attribute TEXT
)
RETURN INTEGER
```

Table 10-78 PKG_UTIL.SESSION_CLEAR_CONTEXT parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
namespace	TEXT	IN	No	Namespace of an attribute.
client_identifier	TEXT	IN	No	Usually the value of this parameter is the same as that of **namespace**. If this parameter is set to **null**, all namespaces are modified by default.
attribute	TEXT	IN	No	Attribute to be cleared.

- PKG_UTIL.SESSION_SEARCH_CONTEXT

  Searches for an attribute value.

  The prototype of the PKG_UTIL.SESSION_SEARCH_CONTEXT function is as follows:

  ```
  PKG_UTIL.SESSION_SEARCH_CONTEXT(
  namespace TEXT,
  attribute TEXT
  )
  RETURN INTEGER
  ```


Table 10-79 PKG_UTIL.SESSION_SEARCH_CONTEXT parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
namespace	TEXT	IN	No	Namespace of an attribute.
attribute	TEXT	IN	No	Attribute value to be searched for.

- PKG_UTIL.SESSION_SET_CONTEXT

  Sets the attribute value.

  The prototype of the PKG_UTIL.SESSION_SET_CONTEXT function is as follows:

```
PKG_UTIL.SESSION_SET_CONTEXT(
namespace TEXT,
attribute TEXT,
value TEXT
)
RETURN INTEGER
```

Table 10-80 PKG_UTIL.SESSION_SET_CONTEXT parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
namespace	TEXT	IN	No	Namespace of an attribute.
attribute	TEXT	IN	No	Attribute to be set.
value	TEXT	IN	No	Attribute value.

- PKG_UTIL.UTILITY_GET_TIME

  Prints the Unix timestamp.

  The prototype of the PKG_UTIL.UTILITY_GET_TIME function is as follows:

  ```
  PKG_UTIL.UTILITY_GET_TIME()
  RETURN BIGINT
  ```

- PKG_UTIL.UTILITY_FORMAT_ERROR_BACKTRACE

  Displays the error stack of a stored procedure.

  The prototype of the PKG_UTIL.UTILITY_FORMAT_ERROR_BACKTRACE function is as follows:

  ```
  PKG_UTIL.UTILITY_FORMAT_ERROR_BACKTRACE()
  RETURN TEXT
  ```

- PKG_UTIL.UTILITY_FORMAT_ERROR_STACK

  Displays the error information about a stored procedure.

  The prototype of the PKG_UTIL.UTILITY_FORMAT_ERROR_STACK function is as follows:

  ```
  PKG_UTIL.UTILITY_FORMAT_ERROR_STACK()
  RETURN TEXT
  ```

- PKG_UTIL.UTILITY_COMPILE_SCHEMA

  Recompiles packages, functions and stored procedures under the specified schema.

  The prototype of the PKG_UTIL.UTILITY_COMPILE_SCHEMA function is as follows:

  ```
  PKG_UTIL.UTILITY_COMPILE_SCHEMA (
  schema IN VARCHAR2,
  compile_all IN BOOLEAN DEFAULT TRUE,
  reuse_settings IN BOOLEAN DEFAULT FALSE
  )
  RETURNS VOID
  ```

- PKG_UTIL.GS_COMPILE_SCHEMA

  Recompiles packages, functions and stored procedures under the specified schema.

  The prototype of the PKG_UTIL.GS_COMPILE_SCHEMA stored procedure is as follows:

  ```
  PKG_UTIL.GS_COMPILE_SCHEMA (
  schema_name IN VARCHAR2 DEFAULT NULL,
  compile_all IN BOOLEAN DEFAULT FALSE,
  retry_times IN INT DEFAULT 10
  )
  ```


Table 10-81 PKG_UTIL.GS_COMPILE_SCHEMA parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
schema_name	VARCHAR2	IN	Yes	Specifies the name of the namespace.
compile_all	BOOLEAN	IN	Yes	Specifies to compile all objects. • **false**: Compile the packages, functions, and stored procedures whose status is **false** in the **pg_object** table. • **true**: Compile all packages, functions, and stored procedures in the **pg_object** table.
retry_times	INT	IN	Yes	Specifies the number of retry times.

- PKG_UTIL.UTILITY_FORMAT_CALL_STACK

  Displays the call stack of a stored procedure.

  The prototype of the PKG_UTIL.UTILITY_FORMAT_CALL_STACK function is as follows:

  ```
  PKG_UTIL.UTILITY_FORMAT_CALL_STACK()
  RETURN TEXT
  ```

  Example:

  ```
  -- Delete a schema.
  drop schema if exists pkg_var_test cascade;
  -- Create a schema named pkg_var_test.
  create schema pkg_var_test;
  -- Set the schema and parameters.
  set current_schema = pkg_var_test;
  set behavior_compat_options ='plpgsql_dependency';
  -- Create a package.
  create or replace package test_pkg as
      referenced_var int;
      unreferenced_var int;
  end test_pkg;
  /
  -- Create a function.
  create or replace function test_func return int
  is
  begin
    return 1;
  end;
  ```



```
/
-- Create a stored procedure.
create or replace procedure test_proc
is
  proc_var int;
begin
  proc_var := 1;
end;
/
-- Recompile package functions and stored procedures under pkg_var_test.
call pkg_util.utility_compile_schema('pkg_var_test');
Or
call pkg_util.gs_compile_schema(' ',false,2);
-- Delete pkg_var_test.
drop schema if exists pkg_var_test cascade;
```

- PKG_UTIL.APP_SET_MODULE

  Sets the value of a module.

  The prototype of the PKG_UTIL.APP_SET_MODULE stored procedure is as follows:

  ```
  PKG_UTIL.APP_SET_MODULE (
     str TEXT)
  RETURNS VOID
  ```

  Table 10-82 PKG_UTIL.APP_SET_MODULE parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
text	TEXT	IN	Yes	Specifies the value of a module to be set.

  Example:

  ```
  CALL PKG_UTIL.APP_SET_MODULE('set module');
   app_set_module
  ----------------

  (1 row)
  ```

- PKG_UTIL.APP_READ_MODULE

  Reads the value of a module.

  The prototype of the PKG_UTIL.APP_READ_MODULE stored procedure is as follows:

  ```
  PKG_UTIL.APP_READ_MODULE(
  OUT BUFFER TEXT
  )
  ```


Table 10-83 PKG_UTIL.APP_READ_MODULE parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
buffer	TEXT	OUT	Yes	Specifies the value of a module to be returned.

Example:

```
DECLARE
    module VARCHAR2(64);
BEGIN
    PKG_UTIL.APP_READ_MODULE(module);
    DBE_OUTPUT.PRINT_LINE(module);
END;
/
set module
ANONYMOUS BLOCK EXECUTE
```

- PKG_UTIL.APP_SET_ACTION

  Sets the value of an action.

  The prototype of the PKG_UTIL.APP_SET_ACTION stored procedure is as follows:

  ```
  PKG_UTIL.APP_SET_ACTION (
     str TEXT)
  RETURNS VOID
  ```

Table 10-84 PKG_UTIL.APP_SET_ACTION parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
text	TEXT	IN	Yes	Specifies the value of an action to be set.

Example:

```
CALL PKG_UTIL.APP_SET_ACTION('set action');
 app_set_action
----------------


(1 row)
```

- PKG_UTIL.APP_READ_ACTION

  Reads the value of an action.

  The prototype of the PKG_UTIL.APP_READ_ACTION stored procedure is as follows:

  ```
  PKG_UTIL.APP_READ_ACTION(
  OUT buffer TEXT
  )
  ```

  Table 10-85 PKG_UTIL.APP_READ_ACTION parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
buffer	TEXT	OUT	Yes	Specifies the value of an action to be returned.

  Example:

  ```
  DECLARE
      action VARCHAR2(64);
  BEGIN
      PKG_UTIL.APP_READ_ACTION(action);
      DBE_OUTPUT.PRINT_LINE(action);
  END;
  /
  set action
  ANONYMOUS BLOCK EXECUTE
  ```

- PKG_UTIL.MODIFY_PACKAGE_STATE

  Modifies the PL/SQL status of the current session.

  The prototype of the PKG_UTIL.MODIFY_PACKAGE_STATE stored procedure is as follows:

  ```
  PKG_UTIL.MODIFY_PACKAGE_STATE (
     flags INT)
  RETURNS VOID
  ```


Table 10-86 PKG_UTIL.MODIFY_PACKAGE_STATE parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
flags	INT	INT	Yes	Bit flag of the operation performed on the PL/SQL. When **flags** is set to **1**, the session cache of the PL/SQL that is currently running in the session is released after the top-level PL/SQL is executed. The current value of any package is cleared globally and the cached cursor is closed. On subsequent use, PL/SQL re-instantiates and re-initializes the package globally. Other bit flags are not supported.

Example:

```
CALL PKG_UTIL.MODIFY_PACKAGE_STATE(1);
 modify_package_state
----------------------

(1 row)
```

10.12.1.3 DBE_DESCRIBE

For details about the basic interfaces supported by DBE_DESCRIBE, see **Table 1 DBE_DESCRIBE basic interfaces**.

Table 10-87 DBE_DESCRIBE basic interfaces

Interface	Description
DBE_DESCRIBE.GET_PROCEDURE_NAME	This is an internal function and is not recommended. The name of a stored procedure or function is extracted from the original input of a user.
DBE_DESCRIBE.IS_NUMBER_TYPE	This is an internal function and is not recommended. It is used to determine whether the data type is numeric.

- DBE_DESCRIBE.GET_PROCEDURE_NAME

  This is an internal function and is not recommended. It is used to extract the name of a stored procedure or function from the original input of a user.

  The prototype of the DBE_DESCRIBE.GET_PROCEDURE_NAME function is as follows:

  ```
  DBE_DESCRIBE.GET_PROCEDURE_NAME(
  name IN VARCHAR2)
  RETURNS VARCHAR2;
  ```

  Table 10-88 DBE_DESCRIBE.GET_PROCEDURE_NAME interface parameters

Parameter	Type	Input/ Output Parameter	Whether NULL Is Allowed	Description
name	VARCHAR2	IN	No	Entity name.

- DBE_DESCRIBE.IS_NUMBER_TYPE

  This is an internal function and is not recommended. It is used to check whether the type is numeric.

  The prototype of the DBE_DESCRIBE.IS_NUMBER_TYPE function is as follows:

  ```
  DBE_DESCRIBE.IS_NUMBER_TYPE(
  type_oid IN INTEGER)
  RETURNS BOOLEAN;
  ```

  Table 10-89 DBE_DESCRIBE.IS_NUMBER_TYPE interface parameters

Parameter	Type	Input/Output Parameter	Whether NULL Is Allowed	Description
type_oid	INTEGER	IN	Yes	OID of the type.

10.12.1.4 DBE_XML

Table 10-90 lists all APIs supported by DBE_XML.

Table 10-90 DBE_XML parameters

API	Description
DBE_XML.XML_FREE_PARSER	Frees a parser.
DBE_XML.XML_PARSER_GET_DOC	Obtains the parsed document node.
DBE_XML.XML_GET_VALIDATION_MODE	Obtains the validation attribute.

API	Description
DBE_XML.XML_SET_VALIDATION_MODE	Creates a parser instance.
DBE_XML.XML_PARSE_BUFFER	Parses the VARCHAR string.
DBE_XML.XML_PARSE_CLOB	Parses the CLOB string.
DBE_XML.XML_SET_VALIDATION_MODE	Sets the validation attribute.
DBE_XML.XML_DOM_APPEND_CHILD	Adds the newchild node to the end of the parent(n) node and returns the newly added node.
DBE_XML.XML_DOM_CREATE_ELEMENT	Returns the DOMElement object with the specified name.
DBE_XML.XML_DOM_CREATE_ELEMENT_NS	Returns the DOMElement object with the specified name and namespace.
DBE_XML.XML_DOM_CREATE_TEXT_NODE	Creates and returns a DOMText object.
DBE_XML.XML_DOM_FREE_DOCUMENT	Frees a specified XML DOM object.
DBE_XML.XML_DOM_FREE_ELEMENT	Frees a specified XML DOM object.
DBE_XML.XML_DOM_FREE_NODE	Frees a DOMNode node.
DBE_XML.XML_DOM_FREE_NODELIST	Frees a DOMNodeList node.
DBE_XML.XML_DOM_GET_ATTRIBUTE	Obtains the attributes of a specified XML DOM object.
DBE_XML.XML_DOM_GET_ATTRIBUTES	Returns the attribute values of a DOMNode node as a map.
DBE_XML.XML_DOM_GET_CHILD_NODES	Converts several subnodes under a node into a node list.
DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME	Obtains the list of specified subnodes of a specified XML DOM object.
DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME_NS	Obtains the list of subnodes in the specified namespace of a specified XML DOM object.
DBE_XML.XML_DOM_GET_DOCUMENT_ELEMENT	Returns the first subnode of the specified document.

API	Description
DBE_XML.XML_DOM_GET_FIRST_CHILD	Returns the first subnode of a node.
DBE_XML.XML_DOM_GET_LAST_CHILD	Returns the last subnode of a node.
DBE_XML.XML_DOM_GET_LENGTH	Returns the number of nodes based on the content in the node of the specified type.
DBE_XML.XML_DOM_GET_LOCALNAME	Returns the local name of the given object.
DBE_XML.XML_DOM_GET_NAMED_ITEM	Returns the node specified by name.
DBE_XML.XML_DOM_GET_NAMED_ITEM_NS	Returns the node specified by name and namespace.
DBE_XML.XML_DOM_GET_NEXT_SIBLING	Returns the next node of the specified node.
DBE_XML.XML_DOM_GET_NODE_NAME	Returns the name of a node.
DBE_XML.XML_DOM_GET_NODE_TYPE	Returns the type of a node.
DBE_XML.XML_DOM_GET_NODE_VALUE	Returns the value of a DOMNode node.
DBE_XML.XML_DOM_GET_PARENT_NODE	Returns the parent node of the given DOMNode node.
DBE_XML.XML_DOM_GET_TAGNAME	Obtains the tag name of a specified XML DOM object.
DBE_XML.XML_DOM_HAS_CHILD_NODES	Checks whether the DOMNode object has any subnode.
DBE_XML.XML_DOM_IMPORT_NODE	Copies a node to another node and mounts the copied node to a specified document.
DBE_XML.XML_DOM_IS_NULL	Checks whether the given object is null.
DBE_XML.XML_DOM_ITEM	Returns the element corresponding to the index in a list or map based on the index.
DBE_XML.XML_DOM_MAKE_ELEMENT	Returns the DOMElement object after conversion.
DBE_XML.XML_DOM_MAKENODE	Converts the given object to the DOMNode type.
DBE_XML.XML_DOM_NEW_DOM_DOCUMENT_EMPTY	Returns a new DOMDocument object.

API	Description
XML_DOM_NEW_DOM_DOCUMENT_CLOB	Returns a new DOMDocument instance object created from the specified CLOB type.
DBE_XML.XML_DOM_NEW_DOCUMENT_XMLTYPE	Returns a new DOMDocument instance object created from the specified XMLType type.
DBE_XML.XML_DOM_SET_ATTRIBUTE	Sets the attributes of a specified XML DOM object.
DBE_XML.XML_DOM_SET_CHARSET	Sets the character set for a DOMDocument object.
DBE_XML.XML_DOM_SET_DOCTYPE	Sets the external DTD of a DOMDocument object.
DBE_XML.XML_DOM_SET_NODE_VALUE	Sets the value of a node in the DOMNode object.
DBE_XML.XML_DOM_WRITE_TO_BUFFER_DOC	Writes the given DOMDocument object to the buffer.
DBE_XML.XML_DOM_WRITE_TO_BUFFER_NODE	Writes the given DOMNode object to the buffer.
DBE_XML.XML_DOM_WRITE_TO_CLOB_DOC	Writes the given DOMDocument object to a CLOB.
DBE_XML.XML_DOM_WRITE_TO_CLOB_NODE	Writes the given DOMNode object to a CLOB.
DBE_XML.XML_DOM_WRITE_TO_FILE_DOC	Writes an XML node to a specified file using the database character set.
DBE_XML.XML_DOM_WRITE_TO_FILE_NODE	Writes an XML node to a specified file using the database character set.
DBE_XML.XML_DOM_GET_SESSION_TREE_NUM	Displays the number of DOM trees of all types in the current session.
DBE_XML.XML_DOM_GET_DOC_TREES_INFO	Displays statistics such as the memory usage and number of nodes of the DOM tree of the document type.
DBE_XML.XML_DOM_GET_DETAIL_DOC_TREE_INFO	Displays the number of nodes of each type for a specific document variable.

- DBE_XML.XML_FREE_PARSER

  Frees a given parser object.

  The stored procedure prototype of DBE_XML.XML_FREE_PARSER is as follows:

  ```
  DBE_XML.XML_FREE_PARSER(
  id IN RAW(13))
  RETURNS VOID;
  ```


Table 10-91 DBE_XML.XML_FREE_PARSER parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	Yes	Parser object

- DBE_XML.XML_PARSER_GET_DOC

  Returns the root node of the DOM tree document constructed by the parser.

  The prototype of the DBE_XML.XML_PARSER_GET_DOC function is as follows:

```
DBE_XML.XML_PARSER_GET_DOC(
id IN RAW(13))
RETURNS RAW(13);
```

Table 10-92 DBE_XML.XML_PARSER_GET_DOC parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	Yes	Parser object

NOTE

- If the DBE_XML.XML_PARSER_GET_DOC function is empty, null is returned.
- If the parser input by the DBE_XML.XML_PARSER_GET_DOC function has not parsed any document, null is returned.

- DBE_XML.XML_GET_VALIDATION_MODE

  Obtains the parsing validation mode of a specified PARSER. If DTD validation is enabled, **TRUE** is returned. Otherwise, **FALSE** is returned.

  The prototype of the DBE_XML.XML_GET_VALIDATION_MODE function is as follows:

```
DBE_XML.XML_GET_VALIDATION_MODE(
id RAW(13))
RETURNS BOOL;
```

Table 10-93 DBE_XML.XML_GET_VALIDATION_MODE parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	Yes	Parser object

- DBE_XML.XML_NEW_PARSER

  Creates a PARSER object and returns a new parser instance.

  The prototype of the DBE_XML.XML_NEW_PARSER function is as follows:

```
DBE_XML.XML_NEW_PARSER()
RETURNS RAW(13);
```


- DBE_XML.XML_PARSE_BUFFER

  Parses XML documents stored in strings.

  The stored procedure prototype of DBE_XML.XML_PARSE_BUFFER is as follows:

```
DBE_XML.XML_PARSE_BUFFER(
id RAW(13),
xmlstr VARCHAR2)
RETURNS VOID;
```

  Table 10-94 DBE_XML.XML_PARSE_BUFFER parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	Yes	Parser object
xmlstr	VARCHAR2	IN	No	A string that stores XML documents

  NOTE

  - The maximum length of a character string that can be parsed by the XML_PARSE_BUFFER function is 32767. If the length exceeds the maximum, an error is reported.
  - Different from the database A, this database supports only UTF-8 in terms of character encoding, and **version** can only be set to **1.0**. If versions 1.0 to 1.9 are parsing, a warning appears but the execution is normal. For versions later than 1.9, an error is reported.
  - DTD validation differences:
    - **!ATTLIST to type (CHECK|check|Check) "Ch..."** reports an error because the default value **"Ch..."** is not an enumerated value in the brackets. However, the database A does not report this error.
    - **<!ENTITY baidu "www.baidu.com">..... &Baidu;&writer** reports an error because the letters are case sensitive. **Baidu** cannot correspond to **baidu**. However, the database A does not report this error.
  - Namespace validation difference: Undeclared namespace tags are parsed. However, the database A reports an error.
  - Difference in parsing XML predefined entities: **'** and **"** are parsed and translated into ' and ". However, predefined entities in database A are not translated into characters.

- DBE_XML.XML_PARSE_CLOB

  Parses XML documents stored in a CLOB.

  The stored procedure prototype of DBE_XML.XML_PARSE_CLOB is as follows:

```
DBE_XML.XML_PARSE_CLOB(
id IN RAW(13),
doc IN CLOB)
RETURNS VOID;
```


Table 10-95 DBE_XML.XML_PARSE_CLOB parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	Yes	Parser object
doc	CLOB	IN	No	A string that stores XML documents

NOTE

- xml_parse_clob cannot parse CLOBs greater than or equal to 2 GB.
- Different from the database A, this database supports only UTF-8 in terms of character encoding, and **version** can only be set to **1.0**. If versions 1.0 to 1.9 are parsing, a warning appears but the execution is normal. For versions later than 1.9, an error is reported.
- DTD validation differences:
  - **!ATTLIST to type (CHECK|check|Check) "Ch..."** reports an error because the default value **"Ch..."** is not an enumerated value in the brackets. However, the database A does not report this error.
  - **<!ENTITY baidu "www.baidu.com">..... &Baidu;&writer** reports an error because the letters are case sensitive. **Baidu** cannot correspond to **baidu**. However, the database A does not report this error.
- Namespace validation difference: Undeclared namespace tags are parsed. However, the database A reports an error.
- Difference in parsing XML predefined entities: **'** and **"** are parsed and translated into ' and ". However, predefined entities in database A are not translated into characters.

- DBE_XML.XML_SET_VALIDATION_MODE

  Sets the parsing validation mode of a specified PARSER.

  The stored procedure prototype of DBE_XML.XML_SET_VALIDATION_MODE is as follows:

```
DBE_XML.XML_SET_VALIDATION_MODE(
id RAW(13),
validate BOOLEAN)
RETURNS VOID;
```

Table 10-96 DBE_XML.XML_SET_VALIDATION_MODE parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	Yes	Parser object

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
validate	BOOLEAN	IN	Yes	Mode to be set: • **TRUE**: DTD validation is enabled. • **FALSE**: DTD validation is disabled.

NOTE

- If the input parameter **validate** of the DBE_XML.XML_SET_VALIDATION_MODE function is null, the parsing validation mode of the parser is not changed.
- By default, the DTD validation is enabled during parser initialization.

- DBE_XML.XML_DOM_APPEND_CHILD

  Adds the newchild node to the end of the parent(n) node and returns the newly added node.

  The stored procedure prototype of DBE_XML.XML_DOM_APPEND_CHILD is as follows:

```
DBE_XML.XML_DOM_APPEND_CHILD(
   parentid IN RAW(13),
   childid IN RAW(13)
)
RETURNS RAW(13);
```

Table 10-97 DBE_XML.XML_DOM_APPEND_CHILD parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
parentId	RAW(13)	IN	No	XML DOM object
childId	RAW(13)	IN	No	XML DOM object

- DBE_XML.XML_DOM_CREATE_ELEMENT

  Returns the DOMElement object with the specified name.

  The prototype of the DBE_XML.XML_DOM_CREATE_ELEMENT function is as follows:

```
DBE_XML.XML_DOM_CREATE_ELEMENT(
   id IN RAW(13),
   tagname IN VARCHAR2
)
RETURNS RAW(13);
```


Table 10-98 DBE_XML.XML_DOM_CREATE_ELEMENT parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object
tagname	VARCHAR2	IN	No	Name of the new DOMElement object

- DBE_XML.XML_DOM_CREATE_ELEMENT_NS

  Returns the DOMElement object with the specified name and namespace.

  The prototype of the DBE_XML.XML_DOM_CREATE_ELEMENT_NS function is as follows:

```
DBE_XML.XML_DOM_CREATE_ELEMENT_NS(
   id IN RAW(13),
   tagname IN VARCHAR2,
   ns IN VARCHAR2
)
RETURNS RAW(13);
```

Table 10-99 DBE_XML.XML_DOM_CREATE_ELEMENT_NS parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object
tagname	VARCHAR2	IN	No	Name of the new DOMElement object
ns	VARCHAR2	IN	No	Namespace

- DBE_XML.XML_DOM_CREATE_TEXT_NODE

  Creates and returns a DOMText object.

  The prototype of the DBE_XML.XML_DOM_CREATE_TEXT_NODE function is as follows:

```
DBE_XML.XML_DOM_CREATE_TEXT_NODE(
   id IN RAW(13),
   data IN VARCHAR2
)
RETURNS RAW(13);
```

Table 10-100 DBE_XML.XML_DOM_CREATE_TEXT_NODE parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
data	VARCHAR2	IN	No	Content of the new DOMText node

- DBE_XML.XML_DOM_FREE_DOCUMENT

  Frees a specified XML DOM object.

  The stored procedure prototype of DBE_XML.XML_DOM_FREE_DOCUMENT is as follows:

```
DBE_XML.XML_DOM_FREE_DOCUMENT(
    id RAW(13)
)
RETURNS VOID;
```

  Table 10-101 DBE_XML.XML_DOM_FREE_DOCUMENT parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object

- DBE_XML.XML_DOM_FREE_ELEMENT

  Frees a specified XML DOM object.

  The stored procedure prototype of DBE_XML.XML_DOM_FREE_ELEMENT is as follows:

```
DBE_XML.XML_DOM_FREE_ELEMENT (
    id RAW(13)
)
RETURNS VOID;
```

  Table 10-102 DBE_XML.XML_DOM_FREE_ELEMENT parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object

- DBE_XML.XML_DOM_FREE_NODE

  Frees a DOMNode node.

  The prototype of the DBE_XML.XML_DOM_FREE_NODE function is as follows:

```
DBE_XML.XML_DOM_FREE_NODE (
    id RAW(13)
)
RETURNS VOID;
```


Table 10-103 DBE_XML.XML_DOM_FREE_NODE parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object

- DBE_XML.XML_DOM_FREE_NODELIST

  Frees a DOMNodeList node.

  The stored procedure prototype of DBE_XML.XML_DOM_FREE_NODELIST is as follows:

```
DBE_XML.XML_DOM_FREE_NODELIST(
   id IN RAW(13)
)
RETURNS VOID
```

Table 10-104 DBE_XML.XML_DOM_FREE_NODELIST parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object

- DBE_XML.XML_DOM_GET_ATTRIBUTE

  Obtains the attributes of a specified XML DOM object.

  The stored procedure prototype of DBE_XML.XML_DOM_GET_ATTRIBUTE is as follows:

```
DBE_XML.XML_DOM_GET_ATTRIBUTE (
   docid   IN  RAW(13),
   name    IN  VARCHAR2
)
RETURNS VARCHAR2;
```

Table 10-105 DBE_XML.XML_DOM_GET_ATTRIBUTE parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
docid	RAW(13)	IN	No	XML DOM object
name	VARCHAR2	IN	No	String

- DBE_XML.XML_DOM_GET_ATTRIBUTES

  Returns the attribute values of a DOMNode node as a map.

  The prototype of the DBE_XML.XML_DOM_GET_ATTRIBUTES function is as follows:


```
DBE_XML.XML_DOM_GET_ATTRIBUTES (
   id RAW(13)
)
RETURNS RAW(13);
```

Table 10-106 DBE_XML.XML_DOM_GET_ATTRIBUTES parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object

- DBE_XML.XML_DOM_GET_CHILD_NODES

  Converts several subnodes under a node into a node list.

  The prototype of the DBE_XML.XML_DOM_GET_CHILD_NODES function is as follows:

```
DBE_XML.XML_DOM_GET_CHILD_NODES(
   id IN RAW(13)
)
RETURNS RAW(13);
```

Table 10-107 DBE_XML.XML_DOM_GET_CHILD_NODES parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object

- DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME

  Obtains the list of specified subnodes of a specified XML DOM object.

  The stored procedure prototype of DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME is as follows:

```
DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME (
   docid   IN  RAW(13),
   name    IN  VARCHAR2
)
RETURNS RAW(13);
```

Table 10-108 DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
docid	RAW(13)	IN	No	XML DOM object
name	VARCHAR2	IN	No	String

- DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME_NS

  Obtains the list of subnodes in the specified namespace of a specified XML DOM object.

  The stored procedure prototype of DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME_NS is as follows:

```
DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME_NS (
   docid   IN  RAW(13),
   name    IN  VARCHAR2,
   ns      IN  VARCHAR2
)
RETURNS RAW(13);
```

  Table 10-109 DBE_XML.XML_DOM_GET_CHILDREN_BY_TAGNAME_NS parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
docid	RAW(13)	IN	No	XML DOM object
name	VARCHAR2	IN	No	String
ns	VARCHAR2	IN	Yes	String

- DBE_XML.XML_DOM_GET_DOCUMENT_ELEMENT

  Returns the first subnode of the specified document.

  The stored procedure prototype of DBE_XML.XML_DOM_GET_DOCUMENT_ELEMENT is as follows:

```
DBE_XML.XML_DOM_GET_DOCUMENT_ELEMENT(
   id  RAW(13)
)
RETURNS RAW(13);
```

  Table 10-110 DBE_XML.XML_DOM_GET_DOCUMENT_ELEMENT parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object

- DBE_XML.XML_DOM_GET_FIRST_CHILD

  Returns the first subnode of a node.

  The prototype of the DBE_XML.XML_DOM_GET_FIRST_CHILD function is as follows:

```
DBE_XML.XML_DOM_GET_FIRST_CHILD(
   id IN RAW(13)
)
RETURNS RAW(13);
```


Table 10-111 DBE_XML.XML_DOM_GET_FIRST_CHILD parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object

- DBE_XML.XML_DOM_GET_LAST_CHILD

  Returns the last subnode of a node.

  The prototype of the DBE_XML.XML_DOM_GET_LAST_CHILD function is as follows:

```
DBE_XML.XML_DOM_GET_LAST_CHILD(
   id IN RAW(13)
)
RETURNS RAW(13);
```

Table 10-112 DBE_XML.XML_DOM_GET_LAST_CHILD parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object

- DBE_XML.XML_DOM_GET_LENGTH

  Returns the number of nodes based on the content in the node of the specified type.

  The stored procedure prototype of DBE_XML.XML_DOM_GET_LENGTH is as follows:

```
DBE_XML.XML_DOM_GET_LENGTH(
   id RAW(13)
)
RETURNS VOID;
```

Table 10-113 DBE_XML.XML_DOM_GET_LENGTH parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	Yes	XML DOM object

- DBE_XML.XML_DOM_GET_LOCALNAME

  Returns the local name of the given object.

  The stored procedure prototype of DBE_XML.XML_DOM_GET_LOCALNAME is as follows:

```
DBE_XML.XML_DOM_GET_LOCALNAME (
   id  RAW(13)
```



```
)
RETURNS VARCHAR2;
```

Table 10-114 DBE_XML.XML_DOM_GET_LOCALNAME parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	Yes	XML DOM object

- DBE_XML.XML_DOM_GET_NAMED_ITEM

  Returns the node specified by name.

  The prototype of the DBE_XML.XML_DOM_GET_NAMED_ITEM function is as follows:

```
DBE_XML.XML_DOM_GET_NAMED_ITEM(
   id IN RAW(13),
   nodeName IN VARCHAR2
)
RETURNS RAW(13);
```

Table 10-115 DBE_XML.XML_DOM_GET_NAMED_ITEM parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object
nodeName	VARCHAR2	IN	No	Name of the element to be retrieved

- DBE_XML.XML_DOM_GET_NAMED_ITEM_NS

  Returns the node specified by name and namespace.

  The prototype of the DBE_XML.XML_DOM_GET_NAMED_ITEM_NS function is as follows:

```
DBE_XML.XML_DOM_GET_NAMED_ITEM_NS(
   id  RAW(13),
   nodename IN VARCHAR2,
   ns IN VARCHAR2
)
RETURNS RAW(13);
```

Table 10-116 DBE_XML.XML_DOM_GET_NAMED_ITEM_NS parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
nodeName	VARCHAR2	IN	No	Name of the element to be retrieved
ns	VARCHAR2	IN	Yes	Namespace

- DBE_XML.XML_DOM_GET_NEXT_SIBLING

  Returns the next node of the specified node.

  The prototype of the DBE_XML.XML_DOM_GET_NEXT_SIBLING function is as follows:

```
DBE_XML.XML_DOM_GET_NEXT_SIBLING(
    id IN RAW(13)
)
RETURNS RAW(13);
```

  Table 10-117 DBE_XML.XML_DOM_GET_NEXT_SIBLING parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object

- DBE_XML.XML_DOM_GET_NODE_NAME

  Returns the name of a node.

  The prototype of the DBE_XML.XML_DOM_GET_NODE_NAME function is as follows:

```
DBE_XML.XML_DOM_GET_NODE_NAME(
    id IN RAW(13)
)
RETURNS VARCHAR2;
```

  Table 10-118 DBE_XML.XML_DOM_GET_NODE_NAME parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object

- DBE_XML.XML_DOM_GET_NODE_TYPE

  Returns the type of a node.

  The prototype of the DBE_XML.XML_DOM_GET_NODE_TYPE function is as follows:


```
DBE_XML.XML_DOM_GET_NODE_TYPE(
   id IN RAW(13)
)
RETURNS INTEGER;
```

Table 10-119 DBE_XML.XML_DOM_GET_NODE_TYPE parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object

- DBE_XML.XML_DOM_GET_NODE_VALUE

  Returns the value of a DOMNode node.

  The stored procedure prototype of DBE_XML.XML_DOM_GET_NODE_VALUE is as follows:

  ```
  DBE_XML.XML_DOM_GET_NODE_VALUE(
  id IN RAW(13))
  RETURNS VARCHAR2;
  ```

  Table 10-120 DBE_XML.XML_DOM_GET_NODE_VALUE parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	Yes	XML DOM object

- DBE_XML.XML_DOM_GET_PARENT_NODE

  Returns the parent node of the given DOMNode node.

  The stored procedure prototype of DBE_XML.XML_DOM_GET_PARENT_NODE is as follows:

  ```
  DBE_XML.XML_DOM_GET_PARENT_NODE(
  id IN RAW(13))
  RETURNS RAW(13);
  ```

  Table 10-121 DBE_XML.XML_DOM_GET_PARENT_NODE parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	Yes	XML DOM object

- DBE_XML.XML_DOM_GET_TAGNAME

  Obtains the tag name of a specified XML DOM object.

  The stored procedure prototype of DBE_XML.XML_DOM_GET_TAGNAME is as follows:


```
DBE_XML.XML_DOM_GET_TAGNAME (
   docid RAW(13)
)
RETURNS VARCHAR2;
```

Table 10-122 DBE_XML.XML_DOM_GET_TAGNAME parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
docid	RAW(13)	IN	Yes	XML DOM object

- DBE_XML.XML_DOM_HAS_CHILD_NODES

  Checks whether the DOMNode object has any subnode.

  The stored procedure prototype of DBE_XML.XML_DOM_HAS_CHILD_NODES is as follows:

  ```
  DBE_XML.XML_DOM_HAS_CHILD_NODES(
  id IN RAW(13))
  RETURNS BOOLEAN
  ```

  Table 10-123 DBE_XML.XML_DOM_HAS_CHILD_NODES parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	Yes	XML DOM object

- DBE_XML.XML_DOM_IMPORT_NODE

  Copies a node to another node and mounts the copied node to a specified document. If the type of the copied node does not belong to the 12 types specified by constants of XML DOM, an exception indicating that the type is not supported is thrown.

  The prototype of the DBE_XML.XML_DOM_IMPORT_NODE function is as follows:

  ```
  DBE_XML.XML_DOM_IMPORT_NODE(
     doc_id IN RAW(13),
     node_id IN RAW(13),
     deep IN BOOLEAN
  )
  RETURNS RAW(13);
  ```


Table 10-124 DBE_XML.XML_DOM_IMPORT_NODE parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
doc_id	RAW(13)	IN	No	Document to which the node is mounted
node_id	RAW(13)	IN	No	Node to be imported
deep	BOOLEAN	IN	No	Specifies whether to perform recursive import. • If the value is **TRUE**, the node and all its subnodes are imported. • If the value is **FALSE**, the node itself is imported.

- DBE_XML.XML_DOM_IS_NULL

  Checks whether the given object is null. If yes, **TRUE** is returned. If no, **FALSE** is returned.

  The prototype of the DBE_XML.XML_DOM_IS_NULL function is as follows:

```
DBE_XML.XML_DOM_IS_NULL (
   id  RAW(13)
)
RETURNS BOOLEAN;
```

Table 10-125 DBE_XML.XML_DOM_IS_NULL parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	Yes	XML DOM object

- DBE_XML.XML_DOM_ITEM

  Returns the element corresponding to the index in a list or map based on the index.

  The prototype of the DBE_XML.XML_DOM_ITEM function is as follows:

```
DBE_XML.XML_DOM_ITEM (
   id IN RAW(13),
   index IN INTEGER
)
RETURNS RAW(13);
```


Table 10-126 DBE_XML.XML_DOM_ITEM parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object
index	INTEGER	IN	No	Index of the element to be retrieved

- DBE_XML.XML_DOM_MAKE_ELEMENT

  Returns the DOMElement object after conversion.

  The stored procedure prototype of DBE_XML.XML_DOM_MAKE_ELEMENT is as follows:

```
DBE_XML.XML_DOM_MAKE_ELEMENT(
id IN RAW(13))
RETURNS RAW(13)
```

Table 10-127 DBE_XML.XML_DOM_MAKE_ELEMENT parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object

- DBE_XML.XML_DOM_MAKENODE

  Converts the given object to the DOMNode type.

  The stored procedure prototype of DBE_XML.XML_DOM_MAKENODE is as follows:

```
DBE_XML.XML_DOM_MAKENODE(
  id    RAW(13)
)
RETURNS DOMNODE;
```

Table 10-128 DBE_XML.XML_DOM_MAKENODE parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	Yes	XML DOM object

- DBE_XML.XML_DOM_NEW_DOM_DOCUMENT_EMPTY

  Returns a new DOMDocument object.

  The prototype of the DBE_XML.XML_DOM_NEW_DOM_DOCUMENT_EMPTY function is as follows:


```
DBE_XML.XML_DOM_NEW_DOM_DOCUMENT_EMPTY()
RETURNS RAW(13);
```

- XML_DOM_NEW_DOM_DOCUMENT_CLOB

  Returns a new DOMDocument instance object created from the specified CLOB type.

  The prototype of the XML_DOM_NEW_DOM_DOCUMENT_CLOB function is as follows:

```
DBE_XML.XML_DOM_NEW_DOM_DOCUMENT_CLOB(
   content IN  CLOB
)
RETURNS RAW(13);
```

  Table 10-129 XML_DOM_NEW_DOM_DOCUMENT_CLOB parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
content	CLOB	IN	No	Specified CLOB type

- DBE_XML.XML_DOM_NEW_DOCUMENT_XMLTYPE

  Returns a new DOMDocument instance object created from the specified XMLType type.

  The prototype of the DBE_XML.XML_DOM_NEW_DOCUMENT_XMLTYPE function is as follows:

```
DBE_XML.XML_DOM_NEW_DOCUMENT_XMLTYPE(
   content IN  CLOB
)
RETURNS RAW(13);
```

  Table 10-130 DBE_XML.XML_DOM_NEW_DOCUMENT_XMLTYPE parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
content	CLOB	IN	No	Specified CLOB type

- DBE_XML.XML_DOM_SET_ATTRIBUTE

  Sets the attributes of a specified XML DOM object.

  The stored procedure prototype of DBE_XML.XML_DOM_SET_ATTRIBUTE is as follows:

```
DBE_XML.XML_DOM_SET_ATTRIBUTE(
   docid   IN  RAW(13),
   name    IN  VARCHAR2,
   value   IN  VARCHAR2
)
RETURNS VOID;
```


Table 10-131 DBE_XML.XML_DOM_SET_ATTRIBUTE parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object
name	VARCHAR2	IN	No	String
value	VARCHAR2	IN	No	String

- DBE_XML.XML_DOM_SET_CHARSET

  Sets the character set for a DOMDocument object.

  The prototype of the DBE_XML.XML_DOM_SET_CHARSET function is as follows:

```
DBE_XML.XML_DOM_SET_CHARSET(
   id      IN RAW(13),
   charset IN VARCHAR2
)
RETURNS VOID;
```

Table 10-132 DBE_XML.XML_DOM_SET_CHARSET parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object
charset	VARCHAR2	IN	No	Character set

- DBE_XML.XML_DOM_SET_DOCTYPE

  Sets the external DTD of a DOMDocument object.

  The prototype of the DBE_XML.XML_DOM_SET_DOCTYPE function is as follows:

```
dbe_xml.xml_dom_set_doctype(
   id          IN  RAW(13),
   dtd_name    IN  VARCHAR2,
   system_id   IN  VARCHAR2,
   public_id   IN  VARCHAR2
)
RETURNS VOID;
```

Table 10-133 DBE_XML.XML_DOM_SET_DOCTYPE parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
dtd_name	VARCHAR2	IN	No	Name of the DOCType to be initialized
system_id	VARCHAR2	IN	No	ID of the system whose DOCType needs to be initialized
public_id	VARCHAR2	IN	No	Public ID of the DOCType to be initialized

- DBE_XML.XML_DOM_SET_NODE_VALUE

  Sets the value of a node in the DOMNode object.

  The stored procedure prototype of DBE_XML.XML_DOM_SET_NODE_VALUE is as follows:

  ```
  DBE_XML.XML_DOM_SET_NODE_VALUE(
  id IN RAW(13),
  node_value IN VARCHAR2)
  RETURNS VOID
  ```

  Table 10-134 DBE_XML.XML_DOM_SET_NODE_VALUE parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object
node_value	VARCHAR2	IN	No	String to be set in the DOMNode object

- DBE_XML.XML_DOM_WRITE_TO_BUFFER_DOC

  Writes the given DOMDocument object to the buffer.

  The stored procedure prototype of DBE_XML.XML_DOM_WRITE_TO_BUFFER_DOC is as follows:

  ```
  DBE_XML.DBE_XML.XML_DOM_WRITE_TO_BUFFER_DOC(
  id IN RAW(13))
  RETURNS VARCHAR2;
  ```


Table 10-135 DBE_XML.XML_DOM_WRITE_TO_BUFFER_DOC parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	Yes	XML DOM object

- DBE_XML.XML_DOM_WRITE_TO_BUFFER_NODE

  Writes the given DOMNode object to the buffer.

  The stored procedure prototype of DBE_XML.XML_DOM_WRITE_TO_BUFFER_NODE is as follows:

```
DBE_XML.DBE_XML.XML_DOM_WRITE_TO_BUFFER_NODE(
id IN RAW(13))
RETURNS VARCHAR2;
```

Table 10-136 DBE_XML.XML_DOM_WRITE_TO_BUFFER_NODE parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	Yes	XML DOM object

- DBE_XML.XML_DOM_WRITE_TO_CLOB_DOC

  Writes the given DOMDocument object to a CLOB.

  The stored procedure prototype of DBE_XML.XML_DOM_WRITE_TO_CLOB_DOC is as follows:

```
DBE_XML.XML_DOM_WRITE_TO_CLOB_DOC(
    id IN RAW(13)
)
RETURNS VARCHAR2;
```

Table 10-137 DBE_XML.XML_DOM_WRITE_TO_CLOB_DOC parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	Yes	XML DOM object

- DBE_XML.XML_DOM_WRITE_TO_CLOB_NODE

  Writes the given DOMNode object to a CLOB.

  The stored procedure prototype of DBE_XML.XML_DOM_WRITE_TO_CLOB_NODE is as follows:

```
DBE_XML.XML_DOM_WRITE_TO_CLOB_NODE(
    id          IN  RAW(13)
)
RETURNS CLOB;
```


Table 10-138 DBE_XML.XML_DOM_WRITE_TO_CLOB_NODE parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	Yes	XML DOM object

- DBE_XML.XML_DOM_WRITE_TO_FILE_DOC

  Writes an XML node to a specified file using the database character set.

  The stored procedure prototype of DBE_XML.XML_DOM_WRITE_TO_FILE_DOC is as follows:

```
DBE_XML.XML_DOM_WRITE_TO_FILE_DOC(
id IN  RAW(13),
file_dir IN  VARCHAR2)
RETURNS VOID

DBE_XML.XML_DOM_WRITE_TO_FILE_DOC(
id IN  RAW(13),
file_dir IN  VARCHAR2,
charset  IN  VARCHAR2)
RETURNS VOID PACKAGE
```

Table 10-139 DBE_XML.XML_DOM_WRITE_TO_FILE_DOC parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	Yes	XML DOM object
file_dir	VARCHAR2	IN	No	File to be written
charset	VARCHAR2	IN	No	Specified character set

- DBE_XML.XML_DOM_WRITE_TO_FILE_NODE

  Writes an XML node to a specified file using the database character set.

  The stored procedure prototype of DBE_XML.XML_DOM_WRITE_TO_FILE_NODE is as follows:

```
DBE_XML.XML_DOM_WRITE_TO_FILE_NODE(
id IN  RAW(13),
filename IN  VARCHAR2)
RETURNS VOID
```


Table 10-140 DBE_XML.XML_DOM_WRITE_TO_FILE_NODE parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object
filename	VARCHAR2	IN	No	Specified file address

- DBE_XML.XML_DOM_GET_SESSION_TREE_NUM

  Queries the number of DOM trees of all types in the current session.

  The prototype of the DBE_XML.XML_DOM_GET_SESSION_TREE_NUM function is as follows:

  ```
  DBE_XML.XML_DOM_GET_SESSION_TREE_NUM()
  RETURNS INTEGER
  ```

- DBE_XML.XML_DOM_GET_DOC_TREES_INFO

  Queries the DOM tree information of the document type in the current session, such as the memory usage.

  The prototype of the DBE_XML.XML_DOM_GET_DOC_TREES_INFO function is as follows:

  ```
  DBE_XML.XML_DOM_GET_DOC_TREES_INFO()
  RETURNS VARCHAR2
  ```

- DBE_XML.XML_DOM_GET_DETAIL_DOC_TREE_INFO

  Queries the number of subnodes of each type in the transferred document.

  The prototype of the DBE_XML.XML_DOM_GET_DETAIL_DOC_TREE_INFO function is as follows:

  ```
  DBE_XML.XML_DOM_GET_DETAIL_DOC_TREE_INFO(
  id IN  RAW(13))
  RETURNS VARCHAR2
  ```

Table 10-141 DBE_XML.XML_DOM_GET_DETAIL_DOC_TREE_INFO parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
id	RAW(13)	IN	No	XML DOM object

10.12.2 Secondary Encapsulation APIs (Recommended)

10.12.2.1 DBE_APPLICATION_INFO

Interface Description

Table 10-142 provides all interfaces supported by the **DBE_APPLICATION_INFO** package. **DBE_APPLICATION_INFO** applies to the current session.

Table 10-142 DBE_APPLICATION_INFO

Interface	Description
DBE_APPLICATION_INFO.SET_CLIENT_INFO	Writes client information.
DBE_APPLICATION_INFO.READ_CLIENT_INFO	Reads client information.
DBE_APPLICATION_INFO.SET_MODULE	Sets the name of the currently running module to the new module. You can set the module and action.
DBE_APPLICATION_INFO.READ_MODULE	Reads the values of the module and action columns of the current session.
DBE_APPLICATION_INFO.SET_ACTION	Sets the name of the current action in the current module.

- DBE_APPLICATION_INFO.SET_CLIENT_INFO

  Writes client information. The prototype of the DBE_APPLICATION_INFO.SET_CLIENT_INFO function is as follows:

```
DBE_APPLICATION_INFO.SET_CLIENT_INFO(
    str text
)returns void;
```

  Table 10-143 DBE_APPLICATION_INFO.SET_CLIENT_INFO interface parameters

Parameter	Description
str	Client information to be written. The maximum length is 64 bytes. If the length exceeds 64 bytes, it will be truncated.

- DBE_APPLICATION_INFO.READ_CLIENT_INFO

  The prototype of the DBE_APPLICATION_INFO.READ_CLIENT_INFO function is as follows:

```
DBE_APPLICATION_INFO.READ_CLIENT_INFO(
 OUT client_info text);
```


Table 10-144 DBE_APPLICATION_INFO.READ_CLIENT_INFO interface parameters

Parameter	Description
client_info	Client information

- DBE_APPLICATION_INFO.SET_MODULE

  Sets the name of the currently running module to the new module. The prototype of the DBE_APPLICATION_INFO.SET_MODULE function is as follows:

```
DBE_APPLICATION_INFO.SET_MODULE(
    IN module_name text,
    IN action_name text
);
```

Table 10-145 DBE_APPLICATION_INFO.SET_MODULE parameters

Parameter	Description
module_name	Name of the running module. The maximum length is 64 bytes. If the length exceeds 64 bytes, it will be truncated.
action_name	Name of the current action in the current module. The maximum length is 64 bytes. If the length exceeds 64 bytes, it will be truncated.

  Example:

```
CALL dbe_application_info.set_module('module_name','action_name');
 set_module
------------


(1 row)
```

- DBE_APPLICATION_INFO.READ_MODULE

  Reads the values of the module and action columns of the current session. The prototype of the DBE_APPLICATION_INFO.READ_MODULE function is as follows:

```
DBE_APPLICATION_INFO.READ_MODULE(
    OUT module_name text,
    OUT action_name text
);
```

Table 10-146 DBE_APPLICATION_INFO.READ_MODULE parameters

Parameter	Description
module_name	Name of the running module.
action_name	Name of the current action in the current module.

Example:

```
DECLARE
 module varchar2(64);
 action varchar2(64);
BEGIN
 dbe_application_info.read_module(module,action);
 dbe_output.print_line(module);
 dbe_output.print_line(action);
END;
/
module_name
action_name
ANONYMOUS BLOCK EXECUTE
```

- DBE_APPLICATION_INFO.SET_ACTION

  Sets the name of the current action in the current module. The prototype of the DBE_APPLICATION_INFO.SET_ACTION function is as follows:

```
DBE_APPLICATION_INFO.SET_ACTION(
      IN action_name text
);
```

Table 10-147 DBE_APPLICATION_INFO.SET_ACTION parameters

Parameter	Description
action_name	Name of the current action in the current module. The maximum length is 64 bytes. If the length exceeds 64 bytes, it will be truncated.

Examples

```
CALL dbe_application_info.set_action('action_name');
 set_action
------------


(1 row)
```

10.12.2.2 DBE_COMPRESSION

Interface Description

Evaluates the sampling compression rate of a specified data object or obtains the compression type of a specified row of data based on the input parameters.

Table 10-148 DBE_COMPRESSION

Interface	Description
GET_COMPRESSION_RATIO	Evaluates the sampling compression ratio of a specified data object based on input parameters.
GET_COMPRESSION_TYPE	Obtains the compression type of data in a specified row based on input parameters.

- DBE_COMPRESSION.GET_COMPRESSION_RATIO

  Evaluates the sampling compression rate of a specified data object based on the input parameters. The prototype is as follows:

```
DBE_COMPRESSION.GET_COMPRESSION_RATIO (
scratchtbsname   IN  TEXT,
ownname          IN  TEXT,
objname          IN  TEXT,
subobjname       IN  TEXT,
comptype         IN  INTEGER,
blkcnt_cmp       OUT INTEGER,
blkcnt_uncmp     OUT INTEGER,
row_cmp          OUT INTEGER,
row_uncmp        OUT INTEGER,
cmp_ratio        OUT NUMBER,
comptype_str     OUT VARCHAR2,
sample_ratio     IN  NUMBER  DEFAULT 20,
objtype          IN  INTEGER DEFAULT 1);
```

Table 10-149 DBE_COMPRESSION.GET_COMPRESSION_RATIO parameters

Parameter	Description
scratchtbsname	Tablespace to which a data object belongs.
ownname	Data object owner (schema to which the data object belongs).
objname	Data object name.
subobjname	Name of a data subobject.
comptype	Compression type. The options are as follows: ● **1**: uncompressed. ● **2**: advanced compression.
blkcnt_cmp	Number of pages occupied by sampled rows after compression.
blkcnt_uncmp	Number of pages occupied by sampled rows before compression.
row_cmp	Number of compressed rows on a single page.
row_uncmp	Number of uncompressed rows on a single page.
cmp_ratio	Compression ratio.
comptype_str	Character string of the compression type.
sample_ratio	Sampling rate.
objtype	Object type. The options are as follows: ● **1**: table object.

- DBE_COMPRESSION.GET_COMPRESSION_TYPE

  This interface is used to obtain the compression type of a specified row based on input parameters. This interface is an O&M interface and does not check the visibility. That is, if the input CTID is a deleted row, this interface still returns the latest status of the current row on the page. The prototype is as follows:

```
DBE_COMPRESSION.GET_COMPRESSION_TYPE (
ownname         IN  TEXT,
tablename   IN  TEXT,
ctid            IN  INTEGER,
subobjname  IN  TEXT    DEFAULT NULL,
comptype        OUT INTEGER);
```

Table 10-150 DBE_COMPRESSION.GET_COMPRESSION_TYPE parameters

Parameter	Description
ownname	Data object owner (schema to which the data object belongs).
tablename	Data object name.
ctid	CTID of the target row.
subobjname	Name of a data subobject.
comptype	Compression type. The options are as follows: ● **1**: uncompressed. ● **2**: advanced compression.

Examples

```
gaussdb=# alter database set ilm = on;
gaussdb=# CREATE user user1 IDENTIFIED BY 'Gauss_zzy123';
gaussdb=# CREATE user user2 IDENTIFIED BY 'Gauss_zzy123';
gaussdb=# SET ROLE user1 PASSWORD 'Gauss_zzy123';
gaussdb=# CREATE TABLE TEST_DATA (ORDER_ID INT, GOODS_NAME TEXT, CREATE_TIME TIMESTAMP)
  ILM ADD POLICY ROW STORE COMPRESS ADVANCED ROW AFTER 1 DAYS OF NO MODIFICATION;
INSERT INTO TEST_DATA VALUES (1,'Snack gift basket A', NOW());

gaussdb=# SELECT DBE_COMPRESSION.GET_COMPRESSION_TYPE('user1', 'test_data', '(0,1)', NULL);
 get_compression_type
----------------------
                    1
(1 row)

DECLARE
o_blkcnt_cmp       integer;
o_blkcnt_uncmp     integer;
o_row_cmp          integer;
o_row_uncmp        integer;
o_cmp_ratio        number;
o_comptype_str     varchar2;
begin
dbe_compression.get_compression_ratio(
    SCRATCHTBSNAME  =>  NULL,
    OWNNAME         =>  'user1',
    OBJNAME         =>  'test_data',
    SUBOBJNAME      =>  NULL,
```



```
    COMPTYPE        =>  2,
    BLKCNT_CMP      =>  o_blkcnt_cmp,
    BLKCNT_UNCMP    =>  o_blkcnt_uncmp,
    ROW_CMP         =>  o_row_cmp,
    ROW_UNCMP       =>  o_row_uncmp,
    CMP_RATIO       =>  o_cmp_ratio,
    COMPTYPE_STR    =>  o_comptype_str,
    SAMPLE_RATIO    =>  100,
    OBJTYPE         =>  1);
RAISE INFO 'Number of blocks used by the compressed sample of the object        : %', o_blkcnt_cmp;
RAISE INFO 'Number of blocks used by the uncompressed sample of the object      : %', o_blkcnt_uncmp;
RAISE INFO 'Number of rows in a block in compressed sample of the object        : %', o_row_cmp;
RAISE INFO 'Number of rows in a block in uncompressed sample of the object      : %', o_row_uncmp;
RAISE INFO 'Estimated Compression Ratio of Sample                               : %', o_cmp_ratio;
RAISE INFO 'Compression Type                                                    : %', o_comptype_str;
end;
/
INFO:  Number of blocks used by the compressed sample of the object        : 0
INFO:  Number of blocks used by the uncompressed sample of the object      : 0
INFO:  Number of rows in a block in compressed sample of the object        : 0
INFO:  Number of rows in a block in uncompressed sample of the object      : 0
INFO:  Estimated Compression Ratio of Sample                               : 1
INFO:  Compression Type                                                    : Compress Advanced
```

10.12.2.3 DBE_FILE

The DBE_FILE package provides the capabilities of reading and writing operating system text files for stored procedures.

Precautions

- DBE_FILE requires that files opened using DBE_FILE.FOPEN be encoded using the database character set. If the opened files are not encoded using the expected character set, an encoding verification error occurs when DBE_FILE.READ_LINE is used to read files. DBE_FILE requires that files opened using DBE_FILE.FOPEN_NCHAR be encoded using the UTF-8 character set. If the opened files are not encoded using the expected character set, an encoding verification error occurs when DBE_FILE.READ_LINE_NCHAR is used to read files.
- When DBE_OUTPUT.PUT_LINE is used to print the result obtained by the DBE_FILE.READ_LINE_NCHAR API, ensure that the UTF-8 character set encoding can be converted to the current database character set encoding. If the preceding conditions are met, the result can be properly output. DBE_OUTPUT.PRINT_LINE does not support this function.
- DBE_FILE requires that the character set encoding of the client be the same as that of the database.
- Assume that the database character set encoding is ASCII, the client character set encoding supports Chinese, and the client calls DBE_FILE.WRITE_NCHAR or DBE_FILE.WRITE_LINE_NCHAR to write Chinese content. If the entered content is in UTF-8 encoding format, the written content may not be encoded in UTF-8 format. An error may be reported when the DBE_FILE.READ_LINE_NCHAR is used.

Data Types

- DBE_FILE.FILE_TYPE

Defines the representation of files in the DBE_FILE package. The fields in DBE_FILE.FILE_TYPE are private fields of the DBE_FILE package. Do not change the field value of the type defined in DBE_FILE.FILE_TYPE.

```
CREATE TYPE DBE_FILE.FILE_TYPE AS(
    id INTEGER,
    datatype INTEGER,
    byte_mode BOOLEAN
);
```

Table 10-151 DBE_FILE.FILE_TYPE columns

Parameter	Description
id	File handle
datatype	File data type (CHAR, NCHAR, or binary). Currently, only CHAR and NCHAR files are supported. For a CHAR file, **1** is returned. For an NCHAR file, **2** is returned.
byte_mode	Indicates that the file is opened in binary mode (**TRUE**) or text mode (**FALSE**).

API Description

Table 10-152 lists all APIs supported by the **DBE_FILE** package.

Table 10-152 DBE_FILE

API	Description
DBE_FILE.OPEN/ DBE_FILE.FOPEN	Opens a file based on the specified directory and file name. The corresponding file handle or the DBE_FILE.FILE_TYPE type object encapsulated with the file handle is returned.
DBE_FILE.IS_CLOSE	Checks whether a file handle is closed.
DBE_FILE.IS_OPEN	Checks whether a file handle is opened.
DBE_FILE.READ_LINE	Reads a line of a specified length from an open file.
DBE_FILE.WRITE	Writes data to the buffer of an open file.
DBE_FILE.NEW_LINE	Writes one or more line terminators to the buffer of an open file.
DBE_FILE.WRITE_LINE	Writes data to the buffer of an open file and automatically appends a line terminator.
DBE_FILE.FORMAT_WRITE	Writes data in a specified format to the buffer of an open file.

API	Description
DBE_FILE.GET_RAW	Reads RAW data of a specified number of bytes from an open file.
DBE_FILE.PUT_RAW	Writes raw data to the buffer of an open file.
DBE_FILE.FLUSH	Writes data in the buffer to a physical file.
DBE_FILE.CLOSE	Closes an open file handle.
DBE_FILE.CLOSE_ALL	Closes all file handles opened in a session.
DBE_FILE.REMOVE	Deletes a disk file based on the specified directory and file name. You must have sufficient permissions to perform this operation.
DBE_FILE.RENAME	Renames a disk file, which is similar to the mv command of Unix.
DBE_FILE.COPY	Copies data in a continuous area to a new file. If **start_line** and **end_line** are omitted, the entire file is copied.
DBE_FILE.GET_ATTR	Reads and returns the attributes of a disk file.
DBE_FILE.SEEK	Adjusts the position of a file pointer forward or backward based on the specified number of bytes.
DBE_FILE.GET_POS	Returns the current offset of a file, in bytes.
DBE_FILE.FOPEN_NCHAR	Opens a file of the NCHAR type based on the specified directory and file name.
DBE_FILE.WRITE_NCHAR	Writes data of the NVARCHAR2 type to the buffer of an open file of the NCHAR type.
DBE_FILE.WRITE_LINE_NCHAR	Writes data of the NVARCHAR2 type to the buffer of an open file of the NCHAR type and automatically appends a line terminator.
DBE_FILE.FORMAT_WRITE_NCHAR	Writes data of the NVARCHAR2 type to the buffer of an open file of the NCHAR type in a specified format.
DBE_FILE.READ_LINE_NCHAR	Reads a line of a specified length from an open file of the NCHAR type.

- DBE_FILE.OPEN/DBE_FILE.FOPEN

  Opens a file. You can specify the maximum line size. A maximum of 50 files can be opened in a session. This function returns a file handle of the INTEGER type. The function of DBE_FILE.FOPEN is similar to that of DBE_FILE.OPEN. It returns an object of the type defined in DBE_FILE.FILE_TYPE.

The prototypes of the DBE_FILE.OPEN and DBE_FILE.FOPEN functions are:

```
DBE_FILE.OPEN(
   dir          IN TEXT,
   file_name    IN TEXT,
   open_mode    IN TEXT,
   max_line_size IN INTEGER DEFAULT 1024)
RETURN INTEGER;

DBE_FILE.FOPEN(
   dir          IN TEXT,
   file_name    IN TEXT,
   open_mode    IN TEXT,
   max_line_size IN INTEGER DEFAULT 1024)
RETURN DBE_FILE.FILE_TYPE;
```

Table 10-153 DBE_FILE.OPEN/DBE_FILE.FOPEN parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
dir	text	IN	No	Directory of a file. It is a string, indicating an object name. **NOTE** • Location of the file directory, which needs to be added to system catalog **PG_DIRECTORY**. If the input path does not match the path in **PG_DIRECTORY**, an error indicating that the path does not exist is reported. • When the GUC parameter **safe_data_path** is enabled, you can only use an advanced package to read and write files in the file path specified by **safe_data_path**.
file_name	text	IN	No	File name with an extension (file type), excluding the path name. A path contained in a file name is ignored in the OPEN function. In Unix, the file name cannot end with the combination of a slash and a dot (/.).

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
open_mode	text	IN	No	File opening mode, including: • **r** (read text) • **w** (write text) • **a** (append text) • **rb** (read byte) • **wb** (write byte) • **ab** (append byte) **NOTE** For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits.
max_line_size	integer	IN	Yes	Maximum number of bytes in each line, including newline characters. The minimum value is **1** and the maximum is **32767**. If this parameter is not specified, the default value **1024** is used.

- DBE_FILE.IS_CLOSE

  Checks whether a file handle is closed. A Boolean value is returned. If an invalid file handle is detected, the **INVALID_FILEHANDLE** exception is thrown.

  The prototype of the DBE_FILE.IS_CLOSE function is as follows:

```
DBE_FILE.IS_CLOSE(
   file IN INTEGER)
RETURN BOOLEAN;

DBE_FILE.IS_CLOSE(
   file IN DBE_FILE.FILE_TYPE)
RETURN BOOLEAN;
```


Table 10-154 DBE_FILE.IS_CLOSE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	integer or dbe_file.file_type	IN	Yes	File handle or DBE_FILE.FILE_TYPE object to be checked. If the value is empty, the DBE_FILE.IS_CLOSE API returns empty.

- DBE_FILE.IS_OPEN

  Checks whether a file handle is opened. A Boolean value is returned. If an invalid file handle is detected, the **INVALID_FILEHANDLE** exception is thrown.

  The prototype of the DBE_FILE.IS_OPEN function is as follows:

```
DBE_FILE.IS_OPEN(
   file IN INTEGER)
RETURN BOOLEAN;

DBE_FILE.IS_OPEN(
   file IN DBE_FILE.FILE_TYPE)
RETURN BOOLEAN;
```

Table 10-155 DBE_FILE.IS_OPEN parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	integer or dbe_file.file_type	IN	Yes	File handle or DBE_FILE.FILE_TYPE object to be checked. If the value is empty, the DBE_FILE.IS_OPEN API returns false.

- DBE_FILE.READ_LINE

  Reads data from an open file and stores the read result to the buffer. It reads data to the end of each line excluding the line terminator, to the end of the file, or to the size specified by the **len** parameter. The length of the data to be read cannot exceed the value of **max_line_size** specified when the file is opened.

  The prototype of the DBE_FILE.READ_LINE stored procedure is as follows:

```
DBE_FILE.READ_LINE(
   file   IN  INTEGER,
   buffer OUT TEXT,
```



```
   len    IN  INTEGER DEFAULT NULL)
RETURN TEXT;

DBE_FILE.READ_LINE(
   file   IN  DBE_FILE.FILE_TYPE,
   buffer OUT TEXT,
   len    IN  INTEGER DEFAULT NULL)
RETURN TEXT;
```

Table 10-156 DBE_FILE.READ_LINE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	integer or dbe_file.file_type	IN	No	File handle opened using OPEN or object of the DBE_FILE.FILE_TYPE type opened using FOPEN. The file must be opened in read mode. Otherwise, the INVALID_OPERATION exception is thrown.
buffer	text	OUT	No	Buffer for receiving data.
len	integer	IN	Yes	Number of bytes read from the file. The default value is **NULL**. If the default value **NULL** is used, **max_line_size** is used to specify the line size.

- DBE_FILE.WRITE

  Writes buffer data to the buffer corresponding to a file. The file must be opened in write mode. This operation does not write a line terminator.

  The prototype of the **DBE_FILE.WRITE** function is as follows:

```
DBE_FILE.WRITE(
   file   IN INTEGER,
   buffer IN TEXT)
RETURN BOOLEAN;

DBE_FILE.WRITE(
   file   IN DBE_FILE.FILE_TYPE,
   buffer IN TEXT)
RETURN VOID;
```


Table 10-157 DBE_FILE.WRITE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	integer or dbe_file.file_type	IN	No	File handle opened using OPEN or object of the DBE_FILE.FILE_TYPE type opened using FOPEN. The file must be opened in write mode. This operation does not write line terminators.
buffer	text	IN	Yes	Text data to be written to the file. The accumulated write length of each line cannot be greater than or equal to the value of **max_line_size** specified when OPEN or FOPEN is used. Otherwise, an error is reported when the file is refreshed. If this parameter is left empty, the API directly returns an empty value. **NOTE** For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits.

- DBE_FILE.NEW_LINE

  Writes one or more line terminators to the buffer corresponding to a file. The line terminators are related to the platform used.

  The prototype of the **DBE_FILE.NEW_LINE** function is as follows:

```
DBE_FILE.NEW_LINE(
   file      IN INTEGER,
   line_nums IN INTEGER DEFAULT 1)
RETURN BOOLEAN;

DBE_FILE.NEW_LINE(
   file      IN DBE_FILE.FILE_TYPE,
   line_nums IN INTEGER DEFAULT 1)
RETURN VOID;
```


Table 10-158 DBE_FILE.NEW_LINE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	integer or dbe_file.file_type	IN	No	File handle opened using OPEN or object of the DBE_FILE.FILE_TYPE type opened using FOPEN.
line_nums	integer	IN	Yes	Number of line terminators written to a file. The default value is **1**. If this parameter is left blank, line terminators are not written.

- DBE_FILE.WRITE_LINE

  Writes buffer data to the buffer corresponding to a file. The file must be opened in write mode. This operation automatically adds a line terminator.

  The prototype of the **DBE_FILE.WRITE_LINE** function is as follows:

```
DBE_FILE.WRITE_LINE(
   file   IN INTEGER,
   buffer IN TEXT,
   flush  IN BOOLEAN DEFAULT FALSE)
RETURN BOOLEAN;

DBE_FILE.WRITE_LINE(
   file   IN DBE_FILE.FILE_TYPE,
   buffer IN TEXT,
   flush  IN BOOLEAN DEFAULT FALSE)
RETURN VOID;
```


Table 10-159 DBE_FILE.WRITE_LINE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	integer or dbe_file.file_type	IN	No	File handle opened using OPEN or object of the DBE_FILE.FILE_TYPE type opened using FOPEN.
buffer	text	IN	Yes	Text data to be written to the file. The length of each line (including the newline character) cannot be greater than the value of **max_line_size** specified when OPEN or FOPEN is executed or the default value. Otherwise, an error is reported when the file is refreshed. **NOTE** For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits.
flush	boolean	IN	Yes	Determines whether to flush the data in the buffer corresponding to the file to the disk after WRITE_LINE. The default value is **FALSE**.

- DBE_FILE.FORMAT_WRITE

  Writes formatted data to the buffer corresponding to an open file. It is a DBE_FILE.WRITE API that allows formatting.

  The prototype of the DBE_FILE.FORMAT_WRITE function is as follows:

```
DBE_FILE.FORMAT_WRITE(
   file   IN INTEGER,
   format IN TEXT,
   arg1   IN TEXT DEFAULT NULL,
   . . .
   arg6   IN TEXT DEFAULT NULL)
RETURN BOOLEAN;

DBE_FILE.FORMAT_WRITE(
   file   IN DBE_FILE.FILE_TYPE,
   format IN TEXT,
   arg1   IN TEXT DEFAULT NULL,
   . . .
   arg6   IN TEXT DEFAULT NULL)
RETURN VOID;
```


Table 10-160 DBE_FILE.FORMAT_WRITE parameters

Parameter	Type	Input / Output Parameter	Can Be Empty	Description
file	integer or dbe_file.file_type	IN	No	File handle opened using OPEN or object of the DBE_FILE.FILE_TYPE type opened using FOPEN.
format	text	IN	Yes	Formatted string, containing the text and format characters \n and %s. If this parameter is left blank, no data is written.
[arg1 . . .arg6]	text	IN	Yes	Six optional parameters. The parameters and the positions of characters to be formatted are in one-to-one correspondence. If the parameter corresponding to a character to be formatted is not provided or the parameter is empty, an empty string is used to replace %s.

- DBE_FILE.GET_RAW

  Reads RAW data from an open file, stores the read result in the buffer, and returns the result from *r*.

  The prototype of the DBE_FILE.GET_RAW stored procedure is as follows:

```
DBE_FILE.GET_RAW(
   file   IN  INTEGER,
   r      OUT RAW,
   length IN  INTEGER DEFAULT NULL)
RETURN RAW;

DBE_FILE.GET_RAW(
   file   IN  DBE_FILE.FILE_TYPE,
   r      OUT RAW,
   length IN  INTEGER DEFAULT NULL)
RETURN BOOLEAN;
```


Table 10-161 DBE_FILE.GET_RAW parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	INTEGER or dbe_file.file_type	IN	No	File handle opened using OPEN or object of the DBE_FILE.FILE_TYPE type opened using FOPEN.
r	RAW	OUT	No	Buffer for receiving RAW data.
length	INTEGER	IN	Yes	Number of bytes read from the file. The default value is **NULL**. If the value is **NULL**, the maximum length of the RAW type is used to specify the size.

- DBE_FILE.PUT_RAW

  Writes raw data to the buffer corresponding to a file.

  The prototype of the DBE_FILE.PUT_RAW function is as follows:

```
DBE_FILE.PUT_RAW (
   file  IN INTEGER,
   r     IN RAW,
   flush IN BOOLEAN DEFAULT FALSE)
RETURN BOOLEAN;

DBE_FILE.PUT_RAW (
   file  IN DBE_FILE.FILE_TYPE,
   r     IN RAW,
   flush IN BOOLEAN DEFAULT FALSE)
RETURN VOID;
```


Table 10-162 DBE_FILE.PUT_RAW parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
file	INTEGER or dbe_file.file_type	IN	No	File handle opened using OPEN or object of the DBE_FILE.FILE_TYPE type opened using FOPEN.
r	RAW	IN	No	RAW data to be written to a file. **NOTE** For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits.
flush	BOOLEAN	IN	Yes	Determines whether to flush data to disks after PUT_RAW. The default value is **FALSE**.

- DBE_FILE.FLUSH

  Writes data in a buffer to a physical file. The buffer data must have a line terminator. This function can write the data in the buffer to the corresponding physical file in time.

  The prototype of the DBE_FILE.FLUSH function is as follows:

```
DBE_FILE.FLUSH(
   file IN INTEGER)
RETURN VOID;

DBE_FILE.FLUSH(
   file IN DBE_FILE.FILE_TYPE)
RETURN VOID;
```


Table 10-163 DBE_FILE.FLUSH parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
file	INTEGER or dbe_file.file_type	IN	No	File handle opened using OPEN or object of the DBE_FILE.FILE_TYPE type opened using FOPEN.

- DBE_FILE.CLOSE

  Closes an open file handle. When this function is called, if there is cached data to be written, exception information may be received. If the handle is normally closed, **TRUE** is returned.

  The prototype of the DBE_FILE.CLOSE function is as follows:

```
DBE_FILE.CLOSE(
    file IN INTEGER)
RETURN BOOLEAN;

DBE_FILE.CLOSE(
    file IN DBE_FILE.FILE_TYPE)
RETURN BOOLEAN;
```

Table 10-164 DBE_FILE.CLOSE parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
file	integer or dbe_file.file_type	IN	No	File handle opened using OPEN or object of the DBE_FILE.FILE_TYPE type opened using FOPEN.

- DBE_FILE.CLOSE_ALL

  Closes all file handles opened in a session. This function can be used for emergency cleanup.

The prototype of the DBE_FILE.CLOSE_ALL function is as follows:

```
DBE_FILE.CLOSE_ALL()
RETRUN VOID;
```

- DBE_FILE.REMOVE

  Deletes a disk file. To use this function, you must have the required permission.

  The prototype of the DBE_FILE.REMOVE function is as follows:

```
DBE_FILE.REMOVE(
   dir       IN TEXT,
   file_name IN TEXT)
RETURN VOID;
```

Table 10-165 DBE_FILE.REMOVE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
dir	TEXT	IN	No	Directory of a file. It is a string, indicating an object name. **NOTE** • Location of the file directory, which needs to be added to system catalog **PG_DIRECTORY**. If the input path does not match the path in **PG_DIRECTORY**, an error indicating that the path does not exist is reported. • When the GUC parameter **safe_data_path** is enabled, you can only use an advanced package to operate files in the file path specified by **safe_data_path**.
file_name	TEXT	IN	No	File name.

- DBE_FILE.RENAME

  Renames a disk file. This function is similar to the mv command of Unix.

  The prototype of the DBE_FILE.RENAME function is as follows:

```
DBE_FILE.RENAME(
   src_dir        IN TEXT,
   src_file_name  IN TEXT,
   dest_dir       IN TEXT,
   dest_file_name IN TEXT,
   overwrite      IN BOOLEAN DEFAULT FALSE)
RETURN VOID;
```


Table 10-166 DBE_FILE.RENAME parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
src_dir	TEXT	IN	No	Directory of the source file (case-sensitive) **NOTE** • Location of the file directory, which needs to be added to system catalog **PG_DIRECTORY**. If the input path does not match the path in **PG_DIRECTORY**, an error indicating that the path does not exist is reported. • When the GUC parameter **safe_data_path** is enabled, you can only use an advanced package to operate files in the file path specified by **safe_data_path**.
src_file_name	TEXT	IN	No	Source file to be renamed
dest_dir	TEXT	IN	No	Target directory (case-sensitive). **NOTE** • Location of the file directory, which needs to be added to system catalog **PG_DIRECTORY**. If the input path does not match the path in **PG_DIRECTORY**, an error indicating that the path does not exist is reported. • When the GUC parameter **safe_data_path** is enabled, you can only use an advanced package to operate files in the file path specified by **safe_data_path**.
dest_file_name	text	IN	No	New file name
overwrite	boolean	IN	Yes	Whether to overwrite the file. If the parameter is left empty or not specified, the parameter is not overwritten. If no rewriting is performed and a file with the same name already exists in the destination directory, an error is reported.

- DBE_FILE.COPY

  Copies data in a continuous area to a new file. If **start_line** and **end_line** are omitted, the entire file is copied.

  The prototype of the DBE_FILE.COPY function is as follows:


```
DBE_FILE.COPY(
   src_dir        IN TEXT,
   src_file_name  IN TEXT,
   dest_dir       IN TEXT,
   dest_file_name IN TEXT,
   start_line     IN INTEGER DEFAULT 1,
   end_line       IN INTEGER DEFAULT NULL)
RETURN VOID;
```

Table 10-167 DBE_FILE.COPY parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
src_dir	TEXT	IN	No	Directory of the source file **NOTE** • Location of the file directory, which needs to be added to system catalog **PG_DIRECTORY**. If the input path does not match the path in **PG_DIRECTORY**, an error indicating that the path does not exist is reported. • When the GUC parameter **safe_data_path** is enabled, you can only use an advanced package to operate files in the file path specified by **safe_data_path**.
src_file_name	TEXT	IN	No	Name of the source file to be copied.
dest_dir	TEXT	IN	No	Directory of the destination file **NOTE** • Location of the file directory, which needs to be added to system catalog **PG_DIRECTORY**. If the input path does not match the path in **PG_DIRECTORY**, an error indicating that the path does not exist is reported. • When the GUC parameter **safe_data_path** is enabled, you can only use an advanced package to operate files in the file path specified by **safe_data_path**.
dest_file_name	TEXT	IN	No	Target file **NOTE** For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits.
start_line	TEXT	IN	No	Number of the line where the copy starts. The default value is **1**.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
end_line	TEXT	IN	Yes	Number of the line where the copy ends. The default value is **NULL**, indicating the end of the file.

- DBE_FILE.GET_ATTR

  Reads and returns the attributes of a disk file.

  The prototype of the DBE_FILE.GET_ATTR stored procedure is as follows:

```
DBE_FILE.GET_ATTR(
    location    IN  TEXT,
    filename    IN  TEXT,
    fexists     OUT BOOLEAN,
    file_length OUT BIGINT,
    block_size  OUT INTEGER)
RETURN RECORD;
```

Table 10-168 DBE_FILE.GET_ATTR parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
location	TEXT	IN	No	File directory **NOTE** • Location of the file directory, which needs to be added to system catalog **PG_DIRECTORY**. If the input path does not match the path in **PG_DIRECTORY**, an error indicating that the path does not exist is reported. • When the GUC parameter **safe_data_path** is enabled, you can only use an advanced package to operate files in the file path specified by **safe_data_path**.
filename	TEXT	IN	No	File name.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
fexists	boolean	OUT	No	Whether the file exists.
file_length	BIGINT	OUT	No	File length (unit: byte). If the file does not exist, **NULL** is returned.
block_size	INTEGE	OUT	No	Block size of the file system (unit: byte). If the file does not exist, **NULL** is returned.

- DBE_FILE.SEEK

  Adjusts the position of a file pointer forward or backward based on the specified number of bytes.

  The prototype of the **DBE_FILE.SEEK** function is as follows:

```
DBE_FILE.SEEK(
   file          IN INTEGER,
   absolute_start IN BIGINT DEFAULT NULL,
   relative_start IN BIGINT DEFAULT NULL)
RETURN VOID;

DBE_FILE.SEEK(
   file          IN DBE_FILE.FILE_TYPE,
   absolute_start IN BIGINT DEFAULT NULL,
   relative_start IN BIGINT DEFAULT NULL)
RETURN VOID;
```


Table 10-169 DBE_FILE.SEEK parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
file	INTEGER or dbe_file.file_type	IN	No	File handle opened using OPEN or object of the DBE_FILE.FILE_TYPE type opened using FOPEN.
absolute_start	BIGINT	IN	Yes	Absolute offset of a file. The default value is **NULL**.
relative_start	BIGINT	IN	Yes	Relative offset of a file. A positive number indicates forward offset and a negative number indicates backward offset. The default value is **NULL**. If both **absolute_start** and this parameter are specified, the **absolute_start** parameter is used.

- DBE_FILE.GET_POS

  Returns the current offset of the file in bytes.

  The prototype of the DBE_FILE.FGETPOS function is as follows:

```
DBE_FILE.GET_POS(
   file IN INTEGER)
RETURN BIGINT;

DBE_FILE.GET_POS(
   file IN DBE_FILE.FILE_TYPE)
RETURN BIGINT;
```


Table 10-170 DBE_FILE.GET_POS parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	INTEGER or dbe_file.file_type	IN	No	File handle opened using OPEN or object of the DBE_FILE.FILE_TYPE type opened using FOPEN.

- DBE_FILE.FOPEN_NCHAR

  Opens a file. You can specify the maximum line size. A maximum of 50 files can be opened in a session. This function returns a DBE_FILE.FILE_TYPE type object that encapsulates a file handle. This function opens a file in national character set mode for input or output.

  The prototype of the DBE_FILE.FOPEN_NCHAR function is as follows:

```
DBE_FILE.FOPEN_NCHAR(
   dir           IN TEXT,
   file_name     IN TEXT,
   open_mode     IN TEXT,
   max_line_size IN INTEGER DEFAULT 1024)
RETURN DBE_FILE.FILE_TYPE;
```


Table 10-171 DBE_FILE.FOPEN_NCHAR parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
dir	TEXT	IN	No	Directory of a file. It is a string, indicating an object name. **NOTE** • Location of the file directory, which needs to be added to system catalog **PG_DIRECTORY**. If the input path does not match the path in **PG_DIRECTORY**, an error indicating that the path does not exist is reported. • When the GUC parameter **safe_data_path** is enabled, you can only use an advanced package to read and write files in the file path specified by **safe_data_path**.
file_name	TEXT	IN	No	File name with an extension (file type), excluding the path name. A path contained in a file name is ignored in the FOPEN_NCHAR function. In Unix, the file name cannot end with the combination of a slash and a dot (/.).
open_mode	TEXT	IN	No	File opening mode, including: • **r** (read text) • **w** (write text) • **a** (append text) • **rb** (read byte) • **wb** (write byte) • **ab** (append byte) **NOTE** For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits.
max_line_size	integer	IN	Yes	Maximum number of bytes in each line, including newline characters. The minimum value is **1** and the maximum is **32767**. If this parameter is not specified, the default value **1024** is used.

- DBE_FILE.WRITE_NCHAR

Writes buffer data to the buffer of a file. The file must be opened in national character set or write mode. This operation does not write a line terminator. The text string is written in the UTF-8 character set format.

The prototype of the DBE_FILE.WRITE_NCHAR function is as follows:

```
DBE_FILE.WRITE_NCHAR(
   file   IN DBE_FILE.FILE_TYPE,
   buffer IN NVARCHAR2)
RETURN VOID;
```

Table 10-172 DBE_FILE.WRITE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	dbe_file.file_type	IN	No	Object of the DBE_FILE.FILE_TYPE type opened using FOPEN_NCHAR. The file must be opened in write mode. This operation does not write line terminators.
buffer	NVARCHAR2	IN	Yes	Text data to be written to the file. The accumulated write length of each line cannot be greater than or equal to the value of **max_line_size** specified by **FOPEN_NCHAR**. Otherwise, an error is reported when the file is refreshed. **NOTE** For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits.

- DBE_FILE.WRITE_LINE_NCHAR

  Writes buffer data to the buffer of a file. The file must be opened in national character set or write mode. This operation automatically adds a line terminator. The text string is written in the UTF8 character set format.

  The prototype of the DBE_FILE.WRITE_LINE_NCHAR function is as follows:

```
DBE_FILE.WRITE_LINE_NCHAR(
   file   IN DBE_FILE.FILE_TYPE,
   buffer IN NVARCHAR2)
RETURN VOID;
```


Table 10-173 DBE_FILE.WRITE_LINE parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
file	dbe_file.file_type	IN	No	Object of the DBE_FILE.FILE_TYPE type opened using FOPEN_NCHAR.
buffer	NVARCHAR2	IN	Yes	Text data to be written to the file. The length of each line (including the newline character) cannot be greater than the value of **max_line_size** specified by **FOPEN_NCHAR** or the default value. Otherwise, an error is reported when the file is refreshed. **NOTE** For the write operation, the system checks the file type. If the ELF file is written, an error is reported and the system exits.

- DBE_FILE.FORMAT_WRITE_NCHAR

  Writes formatted data to the buffer of an open file. It is a DBE_FILE.WRITE_NCHAR API that allows formatting.

  The prototype of the DBE_FILE.FORMAT_WRITE_NCHAR function is as follows:

```
DBE_FILE.FORMAT_WRITE_NCHAR(
   file   IN DBE_FILE.FILE_TYPE,
   format IN NVARCHAR2,
   arg1   IN NVARCHAR2 DEFAULT NULL,
   . . .
   arg5   IN NVARCHAR2 DEFAULT NULL)
RETURN VOID;
```


Table 10-174 DBE_FILE.FORMAT_WRITE_NCHAR parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	dbe_file.file_type	IN	No	Object of the DBE_FILE.FILE_TYPE type opened using FOPEN_NCHAR.
format	NVARCHAR2	IN	Yes	Formatted string, containing the text and format characters \n and %s.
[arg1. . .arg5]	NVARCHAR2	IN	Yes	Five optional parameters. The parameters and the positions of characters to be formatted are in one-to-one correspondence. If the parameter corresponding to a character to be formatted is not provided, an empty string is used to replace %s.

- DBE_FILE.READ_LINE_NCHAR

  Reads data from an open file and stores the read result to the buffer. It reads data to the end of each line excluding the line terminator, to the end of the file, or to the size specified by the **len** parameter. The length of the data to be read cannot exceed the value of **max_line_size** specified by **FOPEN_NCHAR**.

  The prototype of the DBE_FILE.READ_LINE_NCHAR stored procedure is as follows:

```
DBE_FILE.READ_LINE_NCHAR(
   file   IN  DBE_FILE.FILE_TYPE,
   buffer OUT NVARCHAR2,
   len    IN  INTEGER DEFAULT NULL)
RETURN NVARCHAR2;
```


Table 10-175 DBE_FILE.READ_LINE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
file	dbe_file.file_type	IN	No	Object of the DBE_FILE.FILE_TYPE type opened using FOPEN_NCHAR. The file must be opened in read mode. Otherwise, the INVALID_OPERATION exception is thrown.
buffer	NVARCHAR2	OUT	No	Buffer for receiving data.
len	INTEGER	IN	Yes	Number of bytes read from the file. The default value is **NULL**. If the default value **NULL** is used, **max_line_size** is used to specify the line size.

Examples

```
-- Add the /tmp/ directory to the PG_DIRECTORY system catalog as a system administrator.
CREATE OR REPLACE DIRECTORY dir AS '/tmp/';
-- Execution result:
CREATE DIRECTORY
-- Use the DBE_FILE advanced package.
DECLARE
    f INTEGER;
    buffer VARCHAR2;
    raw_buffer RAW;

    f1 DBE_FILE.FILE_TYPE;
    f2 DBE_FILE.FILE_TYPE;

    fexists BOOLEAN;
    file_length BIGINT;
    block_size INTEGER;
    pos BIGINT;

    nvarchar_buffer nvarchar2;
    f_nchar DBE_FILE.FILE_TYPE;
BEGIN
    -- Open a file.
    f := DBE_FILE.OPEN('dir', 'sample.txt', 'w');

    IF DBE_FILE.IS_OPEN(f) = true THEN
        DBE_OUTPUT.PRINT_LINE('file opened');
    END IF;

    -- Close the file.
    DBE_FILE.CLOSE(f);
```



```
    IF DBE_FILE.IS_CLOSE(f) = true THEN
        DBE_OUTPUT.PRINT_LINE('file closed');
    END IF;

    f := DBE_FILE.OPEN('dir', 'sample.txt', 'w');

-- Write a file.
    DBE_FILE.WRITE(f, 'A');
    DBE_FILE.NEW_LINE(f);
    DBE_FILE.WRITE(f, 'B');
    DBE_FILE.WRITE(f, 'C');
    DBE_FILE.NEW_LINE(f, 2);
    DBE_FILE.WRITE_LINE(f, 'ABC');
    DBE_FILE.FORMAT_WRITE(f, '[1 -> %s, 2 -> %s]\n', 'GaussDB', 'DBE_FILE');
    DBE_FILE.PUT_RAW(f, '414243');
    DBE_FILE.NEW_LINE(f);
    DBE_FILE.CLOSE(f);

    -- Create sample_copy.txt and copy the content of sample.txt.
    DBE_FILE.COPY('dir', 'sample.txt', 'dir', 'sample_copy.txt');

    -- Open a file in read mode.
    f := DBE_FILE.OPEN('dir', 'sample_copy.txt', 'r');
-- Read a file.
    DBE_FILE.READ_LINE(f, buffer); -- A
    DBE_OUTPUT.PRINT_LINE(buffer);
    DBE_FILE.READ_LINE(f, buffer); -- BC
    DBE_OUTPUT.PRINT_LINE(buffer);
    DBE_FILE.READ_LINE(f, buffer);
    DBE_FILE.READ_LINE(f, buffer); -- ABC
    DBE_OUTPUT.PRINT_LINE(buffer);
    DBE_FILE.READ_LINE(f, buffer); -- [1 -> GaussDB, 2 -> DBE_FILE]
    DBE_OUTPUT.PRINT_LINE(buffer);
    DBE_FILE.READ_LINE(f, buffer); -- RAW 414243 --> ABC
    DBE_OUTPUT.PRINT_LINE(buffer);

    -- Close the file.
    DBE_FILE.CLOSE(f);

    f1 := DBE_FILE.FOPEN('dir', 'sample1.txt', 'w');
    f2 := DBE_FILE.FOPEN('dir', 'sample2.txt', 'w');
    DBE_FILE.CLOSE_ALL();

    IF DBE_FILE.IS_CLOSE(f1) = true and DBE_FILE.IS_CLOSE(f2) = true THEN
        DBE_OUTPUT.PRINT_LINE('f1 and f2 all closed');
    END IF;

    -- Delete the file.
    DBE_FILE.REMOVE('dir', 'sample1.txt');
    DBE_FILE.REMOVE('dir', 'sample2.txt');
    DBE_FILE.REMOVE('dir', 'sample_copy.txt');

    -- Open a file and clear the data in sample.txt.
    f := DBE_FILE.OPEN('dir', 'sample.txt', 'w');
    DBE_FILE.WRITE_LINE(f, 'ABC');
    DBE_FILE.CLOSE(f);

    f := DBE_FILE.OPEN('dir', 'sample.txt', 'r');
    -- GET_RAW
    DBE_FILE.GET_RAW(f, raw_buffer); -- 0A of 4142430A is a newline character.
    DBE_OUTPUT.PRINT_LINE(raw_buffer);
    DBE_FILE.CLOSE(f);

    -- Obtain file attributes.
    DBE_FILE.GET_ATTR('dir', 'sample.txt', fexists, file_length, block_size);

    IF fexists = true THEN
        DBE_OUTPUT.PRINT_LINE('file length: ' || file_length);
```



```
    END IF;

    -- Change the file name.
    DBE_FILE.RENAME('dir', 'sample.txt', 'dir', 'sample_rename.txt', true);
    f1 := DBE_FILE.FOPEN('dir', 'sample_rename.txt', 'r');
    DBE_FILE.SEEK(f1, 1, null);
    pos := DBE_FILE.GET_POS(f1);
    DBE_OUTPUT.PRINT_LINE('position is: ' || pos);
    DBE_FILE.READ_LINE(f1, buffer); -- BC
    DBE_OUTPUT.PRINT_LINE(buffer);
    DBE_FILE.CLOSE(f1);

    -- FLUSH
    f1 := DBE_FILE.FOPEN('dir', 'sample_rename.txt', 'w');
    DBE_FILE.WRITE_LINE(f1, 'ABCEFG');
    DBE_FILE.FLUSH(f1);

    f2 := DBE_FILE.FOPEN('dir', 'sample_rename.txt', 'r');
    DBE_FILE.READ_LINE(f2, buffer); -- ABCEFG
    DBE_OUTPUT.PRINT_LINE(buffer);
    DBE_FILE.CLOSE(f1);
    DBE_FILE.CLOSE(f2);
    DBE_FILE.REMOVE('dir', 'sample_rename.txt');

    -- NCHAR function
    f_nchar := DBE_FILE.FOPEN_NCHAR('dir', 'sample_nchar.txt', 'w');
    DBE_FILE.WRITE_NCHAR(f_nchar, 'ABCDE');
    DBE_FILE.WRITE_LINE_NCHAR(f_nchar, 'ABCDE');
    DBE_FILE.FORMAT_WRITE_NCHAR(f_nchar, '%s, %s', 'hello', 'world');
    DBE_FILE.CLOSE(f_nchar);
    f_nchar := DBE_FILE.FOPEN_NCHAR('dir', 'sample_nchar.txt', 'r');
    DBE_FILE.READ_LINE_NCHAR(f_nchar, nvarchar_buffer); -- ABCDEABCDE
    DBE_OUTPUT.PRINT_LINE(nvarchar_buffer);
    DBE_FILE.READ_LINE_NCHAR(f_nchar, nvarchar_buffer); -- hello, world
    DBE_OUTPUT.PRINT_LINE(nvarchar_buffer);
    DBE_FILE.CLOSE(f_nchar);
    DBE_FILE.REMOVE('dir', 'sample_nchar.txt');
END;
/

-- Execution result:
file opened
file closed
A
BC
ABC
[1 -> GaussDB, 2 -> DBE_FILE]
ABC
f1 and f2 all closed
4142430A
file length: 4
position is: 1
BC
ABCEFG
ABCDEABCDE
hello, world
ANONYMOUS BLOCK EXECUTE
```

10.12.2.4 DBE_HEAT_MAP

Interface Description

Returns information such as the last modification time of a row in the target data block based on the input parameters. The information is used to browse the basis for determining cold and hot rows. This interface is an O&M interface and has no visibility check. That is, if the input CTID is a deleted row, this interface still returns the latest status of the current row on the page.

Table 10-176 DBE_HEAT_MAP

Interface	Description
ROW_HEAT_MAP	Obtains information such as the last modification time of a row based on the schema to which the object belongs, data object name, data object partition name, and CTID.

- DBE_HEAT_MAP.ROW_HEAT_MAP

  Obtains information such as the last modification time of a row based on the schema to which the object belongs, data object name, data object partition name, and CTID. The prototype is as follows:

```
DBE_HEAT_MAP.ROW_HEAT_MAP(
owner IN VARCHAR2,
segment_name IN VARCHAR2,
partition_name IN VARCHAR2 DEFAULT NULL,
ctid IN TEXT);
```

Table 10-177 DBE_HEAT_MAP.ROW_HEAT_MAP parameters

Parameter	Description
owner	Schema to which a data object belongs.
segment_name	Data object name.
partition_name	Data object partition name. This parameter is optional. The default value is **NULL**.
ctid	Data row ID.

Examples

```
gaussdb=# ALTER DATABASE set ilm = on;
gaussdb=# CREATE Schema HEAT_MAP_DATA;
gaussdb=# SET current_schema=HEAT_MAP_DATA;

gaussdb=# CREATE TABLESPACE example1 RELATIVE LOCATION 'tablespace1';
gaussdb=# CREATE TABLE HEAT_MAP_DATA.heat_map_table(id INT, value TEXT) TABLESPACE example1;
gaussdb=# INSERT INTO HEAT_MAP_DATA.heat_map_table VALUES (1, 'test_data_row_1');

gaussdb=# SELECT * from DBE_HEAT_MAP.ROW_HEAT_MAP(
    owner          =>  'heat_map_data',
    segment_name  =>  'heat_map_table',
    partition_name  => NULL,
    ctid           =>  '(0,1)');
     owner     |  segment_name  | partition_name | tablespace_name | file_id | relative_fno | ctid  | writetime
---------------+----------------+----------------+-----------------+---------+--------------+-------+-----------
 heat_map_data | heat_map_table |                | example1        |   17291 |        17291 | (0,1) |
(1 row)
```


10.12.2.5 DBE_ILM

Interface Description

Implements ILM policies, ILM policy evaluation, and interface for stopping compression jobs.

Table 10-178 DBE_ILM

Interface	Description
EXECUTE_ILM	Evaluates the specified data and ILM policy based on parameters. If the evaluation is passed, the corresponding compression job is generated.
STOP_ILM	Stops a compression job that is being executed based on parameters.

- DBE_ILM.EXECUTE_ILM

  Evaluates the ILM policy based on parameters. The prototype is as follows:

```
DBE_ILM.EXECUTE_ILM (
schema_name         IN     VARCHAR2,
object_name         IN     VARCHAR2,
task_id             OUT    Oid,
subobject_name      IN     VARCHAR2 DEFAULT NULL,
policy_name         IN     VARCHAR2 DEFAULT ILM_ALL_POLICIES,
execution_mode      IN     NUMBER DEFAULT ILM_EXECUTION_ONLINE);
```

  Table 10-179 DBE_ILM.EXECUTE_ILM parameters

Parameter	Description
schema_name	Schema to which an object belongs.
object_name	Object name.
task_id	Descriptor ID of the generated ADO task.
subobject_name	Name of a data subobject.
policy_name	Policy name. You can query the GS_ADM_ILMOBJECTS view to obtain the policy name. The default value **DBE_ILM.ILM_ALL_POLICIES** indicates all policies on the object.
execution_mode	Execution mode. The online mode (ILM_EXECUTION_ONLINE) or offline mode (ILM_EXECUTION_OFFLINE) is not involved in this phase.

- DBE_ILM.STOP_ILM

Stops an ILM policy that is being executed based on parameters. The prototype is as follows:

```
DBE_ILM.STOP_ILM (
TASK_ID               IN NUMBER DEFAULT -1,
P_DROP_RUNNING_JOBS     IN BOOLEAN DEFAULT FALSE,
P_JOBNAME         IN VARCHAR2 DEFAULT NULL);
```

Table 10-180 DBE_ILM.STOP_ILM parameters

Parameter	Description
TASK_ID	Descriptor ID of an ADO task.
P_DROP_RUNNING_JOBS	Determines whether to stop a task that is being executed. The value **true** indicates that the task is forcibly stopped, and the value **false** indicates that the task is not stopped.
P_JOBNAME	Task name.

NOTE

When there are a large number of concurrent requests and you run DBE_ILM.STOP_ILM, the system may display the message "Resources are busy, please try again later." In this case, try again later.

Examples

```
gaussdb=# ALTER DATABASE set ilm = on;
gaussdb=# CREATE Schema ILM_DATA;
gaussdb=# SET current_schema=ILM_DATA;
BEGIN
    DBE_ILM_ADMIN.DISABLE_ILM();
    DBE_ILM_ADMIN.CUSTOMIZE_ILM(1, 15);
    DBE_ILM_ADMIN.CUSTOMIZE_ILM(2, 30);
    DBE_ILM_ADMIN.CUSTOMIZE_ILM(11, 1);
    DBE_ILM_ADMIN.CUSTOMIZE_ILM(12, 10);
    DBE_ILM_ADMIN.CUSTOMIZE_ILM(13, 1024);
    DBE_ILM_ADMIN.CUSTOMIZE_ILM(14, 240);
    DBE_ILM_ADMIN.ENABLE_ILM();
END;
/
-- 1.1.2 prepare test data
gaussdb=# CREATE SEQUENCE ILM_DATA.ORDER_TABLE_SE_ORDER_ID MINVALUE 1;
gaussdb=# CREATE OR REPLACE PROCEDURE ILM_DATA.ORDER_TABLE_CREATE_DATA(NUM INTEGER) IS
BEGIN
    FOR X IN 1..NUM
        LOOP
            INSERT INTO ORDER_TABLE VALUES(ORDER_TABLE_SE_ORDER_ID.nextval, 'Snack gift basket A',
NOW());
        END LOOP;
    COMMIT;
END;
/
-- 1.1.3 normal procedure
-- 1.1.3.1 evaluate successed - all policy
gaussdb=# CREATE TABLE ILM_DATA.ORDER_TABLE (ORDER_ID INT, GOODS_NAME TEXT, CREATE_TIME
TIMESTAMP)
    WITH (STORAGE_TYPE=ASTORE) ILM ADD POLICY ROW STORE COMPRESS ADVANCED ROW AFTER 1
DAYS OF NO MODIFICATION;
BEGIN
    ILM_DATA.ORDER_TABLE_CREATE_DATA(5);
    PERFORM PG_SLEEP(2);
```



```
END;
/

gaussdb=# SELECT ORDER_ID, DBE_COMPRESSION.GET_COMPRESSION_TYPE('ilm_data', 'order_table',
ctid::text, NULL) FROM ILM_DATA.ORDER_TABLE;
 order_id | get_compression_type
----------+----------------------
        1 |                    1
        2 |                    1
        3 |                    1
        4 |                    1
        5 |                    1
(5 rows)

gaussdb=# SELECT ORDER_ID, DBE_HEAT_MAP.ROW_HEAT_MAP('ilm_data','order_table', NULL, ctid::text)
FROM ILM_DATA.ORDER_TABLE;
 order_id |                row_heat_map
----------+-----------------------------------------------
        1 | (ilm_data,order_table,,,16799,16799,"(0,1)",)
        2 | (ilm_data,order_table,,,16799,16799,"(0,2)",)
        3 | (ilm_data,order_table,,,16799,16799,"(0,3)",)
        4 | (ilm_data,order_table,,,16799,16799,"(0,4)",)
        5 | (ilm_data,order_table,,,16799,16799,"(0,5)",)
(5 rows)

DECLARE
    v_taskid number;
BEGIN
    DBE_ILM.EXECUTE_ILM(OWNER        => 'ilm_data',
                        OBJECT_NAME    => 'order_table',
                        TASK_ID        => v_taskid,
                        SUBOBJECT_NAME => NULL,
                        POLICY_NAME    => 'ALL POLICIES',
                        EXECUTION_MODE => 2);
    RAISE INFO 'Task ID is:%', v_taskid;
END;
/
INFO:  Task ID is:1

gaussdb=# SELECT ORDER_ID, DBE_COMPRESSION.GET_COMPRESSION_TYPE('ilm_data', 'order_table',
ctid::text, NULL) FROM ILM_DATA.ORDER_TABLE;
 order_id | get_compression_type
----------+----------------------
        1 |                    1
        2 |                    1
        3 |                    1
        4 |                    1
        5 |                    1
(5 rows)

gaussdb=# CALL DBE_ILM.STOP_ILM(-1, true, NULL);
 stop_ilm
----------

(1 row)
```

10.12.2.6 DBE_ILM_ADMIN

Interface Description

Implements ILM policies, and controls ADO background scheduling and concurrency control parameters.

Table 10-181 DBE_ILM_ADMIN

Interface	Description
CUSTOMIZE_ILM	Customizes ILM policy attributes based on input parameters.
DISABLE_ILM	Disables background scheduling.
ENABLE_ILM	Enables background scheduling.

NOTE

When there are a large number of concurrent requests and you run DBE_ILM_ADMIN.DISABLE_ILM or DBE_ILM_ADMIN.ENABLE_ILM, the system may display the message "Resources are busy, please try again later." In this case, try again later.

- DBE_ILM_ADMIN.CUSTOMIZE_ILM

  Customizes ILM policy attributes based on input parameters. The prototype is as follows:

```
DBE_ILM_ADMIN.CUSTOMIZE_ILM (
  parameter           IN      int8,
  value               IN      int8);
```

Table 10-182 DBE_ILM_ADMIN.CUSTOMIZE_ILM parameters

Parameter	Description
parameter	Sequence number of a parameter.
value	Parameter value.

Table 10-183 DBE_ILM_ADMIN.CUSTOMIZE_ILM parameters

Parameter ID	Parameter Value	Description
1	EXECUTION_INTERVAL	Specifies the frequency of executing an ADO task, in minutes. The default value is **15**. The value is an integer or floating-point number greater than or equal to 1 and less than or equal to 2147483647. The value is rounded down.
2	RETENTION_TIME	Specifies the retention period of ADO-related history records, in days. The default value is **30**. The value is an integer or floating-point number greater than or equal to 1 and less than or equal to 2147483647. The value is rounded down.

Parameter ID	Parameter Value	Description
7	ENABLE	Specifies the background scheduling status which cannot be modified in this interface. Otherwise, the message "Invalid argument value, ENABLED should be change by calling DBE_ILM_ADMIN.ENABLE_ILM and DBE_ILM_ADMIN.DISABLE_ILM" is displayed. Instead, use disable_ilm() and enable() to modify it.
11	POLICY_TIME	Specifies whether the time unit of ADO is day or second. The time unit second is used only for testing. Possible values are: ● **0**: ILM_POLICY_IN_DAYS (default value) ● **1**: ILM_POLICY_IN_SECONDS
12	ABS_JOBLIMIT	Specifies the maximum number of ADO jobs generated by an ADO task. The value is an integer or floating-point number greater than or equal to 0 and less than or equal to 2147483647. The value is rounded down.
13	JOB_SIZELIMIT	Specifies the maximum number of bytes that can be processed by a single ADO job. The unit is MB. The value is an integer or floating-point number greater than or equal to 1 and less than or equal to 2147483647. The value is rounded down.
14	WIND_DURATION	Specifies the maintenance window duration, in minutes. The default value is 240 minutes (4 hours). The value is an integer greater than or equal to 0 and less than 1440 (24 hours).
15	BLOCK_LIMITS	Specifies the upper limit of the instance-level row-store compression rate. The default value is **40**. The value ranges from 0 to 10000, in block/ms, indicating the maximum number of blocks that can be compressed per millisecond. **0** indicates that the rate is not limited.

- DBE_ILM_ADMIN.DISABLE_ILM

  Disables background scheduling. The prototype is as follows:

  ```
  gaussdb=# DBE_ILM_ADMIN.DISABLE_ILM();
  ```

- DBE_ILM_ADMIN.ENABLE_ILM

  Enables background scheduling. The prototype is as follows:

  ```
  gaussdb=# DBE_ILM_ADMIN.ENABLE_ILM();
  ```

  Note: To make background scheduling take effect, you need to enable the GUC parameter **enable_ilm** on the management and control plane.

Examples

```
gaussdb=# CALL DBE_ILM_ADMIN.CUSTOMIZE_ILM(1, 15);
 customize_ilm
---------------

(1 row)
gaussdb=# select * from gs_adm_ilmparameters;
         name         | value
----------------------+-------
 EXECUTION_INTERVAL |    15
 RETENTION_TIME     |    30
 ENABLED            |     1
 POLICY_TIME        |     0
 ABS_JOBLIMIT       |    10
 JOB_SIZELIMIT      |  1024
 WIND_DURATION      |   240
 BLOCK_LIMITS       |    40
(8 rows)
```

10.12.2.7 DBE_LOB

API Description

Table 10-184 lists all APIs supported by the DBE_LOB package.

NOTE

- In database A, the byte content of the space is 00. However, in GaussDB, the byte content corresponding to the space is ASCII code 32.
- In a centralized environment, the maximum size of a CLOB, BLOB, and BFILE is 32 TB.
- LOBMAXSIZE supports a maximum of 1073741771 bytes.

Table 10-184 DBE_LOB

API	Description
DBE_LOB.GET_LENGTH	Obtains and returns the length of a specified LOB. The object cannot be greater than 2 GB.
DBE_LOB.LOB_GET_LENGTH	Obtains and returns the length of a specified LOB or BFILE object.
DBE_LOB.OPEN	Opens a LOB and returns a LOB descriptor.
DBE_LOB.READ	Loads a part of LOB content to the buffer based on the specified length and initial position offset.
DBE_LOB.LOB_READ	Reads a part of LOB (including BFILE) content to the buffer based on the specified length and initial position offset.
DBE_LOB.WRITE	Copies content in the buffer to a LOB based on the specified length and initial position offset.
DBE_LOB.WRITE_APPEND	Copies content in the buffer to the end part of a LOB based on the specified length.

API	Description
DBE_LOB.LOB_WRITE_APPEND	Copies content in the buffer to the end part of a LOB based on the specified length.
DBE_LOB.COPY	Copies content in a LOB to another LOB based on the specified length and initial position offset.
DBE_LOB.LOB_COPY	Copies content in a LOB to another LOB based on the specified length and initial position offset.
DBE_LOB.ERASE	Deletes content in a LOB (less than or equal to 1 GB) based on the specified length and initial position offset.
DBE_LOB.LOB_ERASE	Deletes content in a LOB based on the specified length and initial position offset.
DBE_LOB.CLOSE	Closes a LOB descriptor.
DBE_LOB.MATCH	Returns the position of the *M*th occurrence of a character string in a LOB.
DBE_LOB.COMPARE	Compares two LOBs (including BFILE objects) or a certain part of two LOBs.
DBE_LOB.SUBSTR	Reads a LOB substring and returns the read substring.
DBE_LOB.LOB_SUBSTR	Reads a LOB or BFILE substring and returns the read substring.
DBE_LOB.STRIP	Truncates the LOB of a specified length. After the execution is complete, the length of the LOB is set to the length specified by the **newlen** parameter.
DBE_LOB.LOB_STRIP	Truncates the LOB of a specified length. After the execution is complete, the length of the LOB is set to the length specified by the **newlen** parameter.
DBE_LOB.CREATE_TEMPORARY	Creates a temporary BLOB or CLOB.
DBE_LOB.APPEND	Adds the content of a LOB to another LOB.
DBE_LOB.LOB_APPEND	Adds the content of a LOB to another LOB.
DBE_LOB.FREETEMPORARY	Deletes a temporary BLOB or CLOB.
DBE_LOB.FILEOPEN	Opens a database BFILE file and returns its file descriptor.
DBE_LOB.FILECLOSE	Closes a BFILE file that is opened by **FILEOPEN**.
DBE_LOB.BFILEOPEN	Opens a database BFILE file.
DBE_LOB.BFILECLOSE	Closes a BFILE file that is opened by **BFILEOPEN**.

API	Description
DBE_LOB.LOADFROMFILE	Loads the database BFILE file of the specified length in the specified location to the BLOB in the specified location.
DBE_LOB.LOADFROMBFILE	Loads the database BFILE file of the specified length in the specified location to the LOB in the specified location.
DBE_LOB.LOADBLOBFROMFILE	Loads an external database file of the specified length in the specified location to a BLOB (less than or equal to 1 GB) in a specified location.
DBE_LOB.LOADBLOBFROMBFILE	Loads the database BFILE file of the specified length in the specified location to the BLOB in the specified location.
DBE_LOB.LOADCLOBFROMFILE	Loads an external database file of the specified length in the specified location to a CLOB (less than or equal to 1 GB) in a specified location.
DBE_LOB.LOADCLOBFROMBFILE	Loads the database BFILE file of the specified length in the specified location to the CLOB in the specified location.
DBE_LOB.CONVERTTOBLOB	Converts a CLOB file to a BLOB file (less than or equal to 1 GB).
DBE_LOB.CONVERTTOCLOB	Converts a BLOB file to a CLOB file (less than or equal to 1 GB).
DBE_LOB.LOB_CONVERTTOBLOB	Converts a CLOB file to a BLOB file.
DBE_LOB.LOB_CONVERTTOCLOB	Converts a BLOB file to a CLOB file.
DBE_LOB.GETCHUNKSIZE	Obtains the maximum size of LOB data that can be stored in the chunk structure in the database.
DBE_LOB.LOB_WRITE	Reads the specified length of the source object from the start position, writes the content to the specified offset position of the target LOB, overrides the original content, and returns the target LOB.
DBE_LOB.BFILENAME	Constructs and returns the DBE_LOB.BFILE object based on the directory and file name.

- DBE_LOB.GET_LENGTH

  Obtains and returns the size of a specified LOB. The maximum size of the object is 2 GB.

  The prototype of the DBE_LOB.GET_LENGTH function is as follows:

```
DBE_LOB.GET_LENGTH (
    blob_obj IN BLOB)
```



```
RETURN INTEGER;

DBE_LOB.GET_LENGTH (
   clob_obj IN CLOB)
RETURN INTEGER;
```

Table 10-185 DBE_LOB.GET_LENGTH parameters

Parameter	Description
blob_obj/ clob_obj	BLOB/CLOB whose length is to be obtained

- DBE_LOB.LOB_GET_LENGTH

  Obtains and returns the length of a specified LOB or BFILE file. The maximum size of the object is 32 TB.

  The prototype of the DBE_LOB.LOB_GET_LENGTH function is as follows:

```
DBE_LOB.LOB_GET_LENGTH (
   blob_obj IN BLOB)
RETURN BIGINT;

DBE_LOB.LOB_GET_LENGTH (
   clob_obj IN CLOB)
RETURN BIGINT;

DBE_LOB.LOB_GET_LENGTH (
   bfile IN DBE_LOB.BFILE)
RETURN BIGINT;
```

Table 10-186 DBE_LOB.LOB_GET_LENGTH parameters

Parameter	Description
blob_obj/ clob_obj/bfile	BLOB/CLOB/BFILE whose length is to be obtained

- DBE_LOB.OPEN

  Opens a LOB and returns a LOB descriptor. This procedure is meaningless and is used only for compatibility.

  The prototype of the DBE_LOB.OPEN function is as follows:

```
DBE_LOB.OPEN (
   lob INOUT BLOB);

DBE_LOB.OPEN (
   lob INOUT CLOB);

DBE_LOB.OPEN (
   bfile     INOUT DBE_LOB.BFILE,
   open_mode IN    TEXT DEFAULT 'null');
```

Table 10-187 DBE_LOB.OPEN parameters

Parameter	Description
lob/bfile	Opened BLOB, CLOB or BFILE.

Parameter	Description
open_mode	Operation mode. Currently, the range is [R,W,A,RB,WB,AB].

- DBE_LOB.READ

  Reads a part of LOB content to the output buffer based on the specified length and initial position offset.

  The prototype of the DBE_LOB.READ function is as follows:

```
DBE_LOB.READ (
   blob_obj IN  BLOB,
   amount   IN  INTEGER,
   off_set  IN  INTEGER,
   out_put  OUT RAW);

DBE_LOB.READ (
   clob_obj IN  CLOB,
   amount   IN  INTEGER,
   off_set  IN  INTEGER,
   out_put  OUT VARCHAR2);
```

  Table 10-188 DBE_LOB.READ parameters

Parameter	Description
blob_obj/ clob_obj	BLOB/CLOB to be read
amount	Length of read content **NOTE** If the length to read is less than 1 or greater than 32767, an error is reported.
off_set	Start position for reading the LOB content, that is, the offset bytes to initial position of the LOB content. If the offset is less than 1 or greater than the LOB length, an error is reported. The initial position is 1.
out_put	Target buffer for storing the read LOB content

- DBE_LOB.LOB_READ

  Reads a part of LOB/BFILE content to the output buffer based on the specified length and initial position offset.

  The prototype of the DBE_LOB.LOB_READ function is as follows:

```
DBE_LOB.LOB_READ(
   blob_obj IN    BLOB,
   amount   INOUT BIGINT,
   off_set  IN    BIGINT,
   out_put  OUT   RAW);

DBE_LOB.LOB_READ(
   clob_obj IN    CLOB,
   amount   INOUT BIGINT,
   off_set  IN    BIGINT,
   out_put  OUT   VARCHAR2);

DBE_LOB.LOB_READ(
```

```
  bfile   IN    DBE_LOB.BFILE,
  amount  INOUT BIGINT,
  off_set IN    BIGINT,
  out_put OUT   RAW);
```

Table 10-189 DBE_LOB.LOB_READ parameters

Parameter	Description
blob_obj/ clob_obj/bfile	BLOB/CLOB/BFILE object (can be greater than 1 GB) to be read
amount	Length to read as the IN parameter, or actual read length as the OUT parameter. **NOTE** If the length to read is less than 1 or greater than 32767, an error is reported.
off_set	Start position for reading the LOB content, that is, the offset bytes to initial position of the LOB content. If the offset is less than 1 or greater than the LOB length, an error is reported. The initial position is 1.
out_put	Target buffer for storing the read LOB content

- DBE_LOB.WRITE

  Writes content in the source to a LOB based on the specified length and initial position.

  The prototype of the DBE_LOB.WRITE function is as follows:

```
DBE_LOB.WRITE (
   blob_obj INOUT BLOB,
   amount   IN    INTEGER,
   off_set  IN    INTEGER,
   source   IN    RAW);

DBE_LOB.WRITE (
   clob_obj INOUT CLOB,
   amount   IN    INTEGER,
   off_set  IN    INTEGER,
   source   IN    VARCHAR2);
```

Table 10-190 DBE_LOB.WRITE parameters

Parameter	Description
blob_obj/ clob_obj	BLOB/CLOB to which the buffer content is written
amount	Length to write, up to 32767 characters. **NOTE** If the length to write is less than 1 or greater than the length of the content to be written, an error is reported.

Parameter	Description
off_set	Start position for writing content to the target LOB, that is, the offset bytes to the initial position of LOB content. **NOTE** If the offset is less than 1 or greater than the value of **LOBMAXSIZE**, an error is reported. The initial position is 1, and the maximum value is the maximum length of the LOB type.
source	Content to be written

- DBE_LOB.WRITE_APPEND

  Writes content in the source object to the end part of a LOB based on the specified length.

  The prototype of the DBE_LOB.WRITE_APPEND function is as follows:

```
DBE_LOB.WRITE_APPEND (
   blob_obj   INOUT BLOB,
   amount     IN    INTEGER,
   source_obj IN    RAW);

DBE_LOB.WRITE_APPEND (
   clob_obj   INOUT CLOB,
   amount     IN    INTEGER,
   source_obj IN    VARCHAR2);
```

  Table 10-191 DBE_LOB.WRITE_APPEND parameters

Parameter	Description
blob_obj/ clob_obj	BLOB/CLOB to which the buffer content is written
amount	Length to write, up to 32767 characters. **NOTE** If the length to write is less than 1 or greater than the length of the content to be written, an error is reported.
source_obj	Content to be written

- DBE_LOB.LOB_WRITE_APPEND

  Writes content in the source object to the end part of a LOB based on the specified length.

  The prototype of the DBE_LOB.LOB_WRITE_APPEND function is as follows:

```
DBE_LOB.LOB_WRITE_APPEND(
   blob_obj    INOUT BLOB,
   amount      IN    INTEGER,
   source_obj  IN    RAW);

DBE_LOB.LOB_WRITE_APPEND (
   clob_obj    INOUT CLOB,
   amount      IN    INTEGER,
   source_obj  IN    VARCHAR2);
```


Table 10-192 DBE_LOB.LOB_WRITE_APPEND parameters

Parameter	Description
blob_obj/ clob_obj	BLOB/CLOB to which the buffer content is written
amount	Length to write, up to 32767 characters. **NOTE** If the length to write is less than 1 or greater than the length of the content to be written, an error is reported.
source_obj	Content to be written

- DBE_LOB.COPY

  Copies content in a LOB to another LOB based on the specified length and initial position offset.

  The prototype of the DBE_LOB.COPY function is as follows:

```
DBE_LOB.COPY (
   dest_lob   INOUT BLOB,
   src_lob    IN    BLOB,
   len        IN    INTEGER,
   dest_start IN    INTEGER DEFAULT 1,
   src_start  IN    INTEGER DEFAULT 1);
```

Table 10-193 DBE_LOB.COPY parameters

Parameter	Description
dest_lob	LOB to which the buffer content is to be pasted.
src_lob	LOB from which the buffer content is to be copied.
len	Length of the copied data.
dest_start	Start position for pasting the buffer content to the target LOB (**dest_lob**), that is, the offset bytes to the initial position of LOB content.
src_start	Start position for copying the buffer content from the source LOB (**src_lob**), that is, the offset bytes to the initial position of LOB content.

- DBE_LOB.LOB_COPY

  Copies content in a LOB to another LOB based on the specified length and initial position offset.

  The prototype of the DBE_LOB.LOB_COPY function is as follows:

```
DBE_LOB.LOB_COPY(
   blob_obj    INOUT BLOB,
   source_obj  IN    BLOB,
   amount      IN    BIGINT,
   dest_offset IN    BIGINT DEFAULT 1,
   src_offset  IN    BIGINT DEFAULT 1);

DBE_LOB.LOB_COPY(
   clob_obj    INOUT CLOB,
   source_obj  IN    CLOB,
```



```
  amount      IN    BIGINT,
  dest_offset IN    BIGINT DEFAULT 1,
  src_offset  IN    BIGINT DEFAULT 1);
```

Table 10-194 DBE_LOB.LOB_COPY parameters

Parameter	Description
blob_obj/ clob_obj	LOB to which the buffer content is to be pasted.
source_obj	LOB from which the buffer content is to be copied.
amount	Length of the copied data. **NOTE** If the length to copy is less than 1 or greater than the value of **LOBMAXSIZE**, an error is reported.
dest_offset	Start position for pasting the buffer content to the target LOB, that is, the offset bytes/characters to the initial position of LOB content. The unit is byte for BLOB and character for CLOB. **NOTE** If the offset is less than 1 or greater than the value of **LOBMAXSIZE**, an error is reported.
src_offset	Start position for copying the content from the source object, that is, the offset bytes to the initial position of LOB content. The unit is byte for BLOB and character for CLOB. **NOTE** If the offset is less than 1, an error is reported.

- DBE_LOB.ERASE

  Deletes the content in a BLOB (not greater than 1 GB) based on the specified length and initial position offset. The bytes of the deleted part in the BLOB are filled with 0.

  The prototype of the DBE_LOB.ERASE function is as follows:

```
DBE_LOB.ERASE (
   blob_obj INOUT BLOB,
   amount   INOUT INTEGER,
   off_set  IN    INTEGER DEFAULT 1);
```

Table 10-195 DBE_LOB.ERASE parameters

Parameter	Description
blob_obj	LOB whose content is to be deleted as the IN parameter, or LOB with specified content deleted as the OUT parameter. If this parameter is left empty, an error is reported.

Parameter	Description
amount	Length (in bytes for BLOBs) to delete as the IN parameter, or actual length deleted as the OUT parameter. **NOTE** If the length to delete is less than 1 or this parameter is left empty, an error is reported.
off_set	Start position from which LOB content is to be deleted, that is, the number of offset bytes to the initial position of LOB content. The value cannot be greater than 1 GB. **NOTE** If the offset is less than 1 or this parameter is left empty, an error is reported.

- DBE_LOB.LOB_ERASE

  Deletes the content in the LOB based on the specified length and initial position offset. The bytes of the deleted part in the BLOB are filled with 0, and the characters of the deleted part in the CLOB are filled with spaces. The LOB can be greater than 1 GB and the maximum size is 32 TB.

  The prototype of the DBE_LOB.LOB_ERASE function is as follows:

```
DBE_LOB.LOB_ERASE (
   blob_obj INOUT BLOB,
   amount   INOUT BIGINT,
   off_set  IN    BIGINT DEFAULT 1);

DBE_LOB.LOB_ERASE (
   clob_obj INOUT CLOB,
   amount   INOUT BIGINT,
   off_set  IN    BIGINT DEFAULT 1);
```

Table 10-196 DBE_LOB.LOB_ERASE parameters

Parameter	Description
blob_obj/ clob_obj	LOB whose content is to be deleted as the IN parameter, or LOB with specified content deleted as the OUT parameter. If this parameter is left empty, an error is reported.
amount	Length (BLOB in bytes and CLOB in characters) to delete as the IN parameter, or actual length deleted as the OUT parameter. **NOTE** If the length to delete is less than 1 or this parameter is left empty, an error is reported.

Parameter	Description
off_set	Start position from which the LOB content is to be deleted, that is, the number of bytes relative to the start position of the BLOB content or the number of characters relative to the start position of the CLOB content. **NOTE** If the offset is less than 1 or this parameter is left empty, an error is reported.

- DBE_LOB.CLOSE

  Closes the LOB descriptor that has been opened.

  The prototype of the DBE_LOB.CLOSE function is as follows:

```
DBE_LOB.CLOSE(
   lob IN BLOB);

DBE_LOB.CLOSE (
   lob IN CLOB);

DBE_LOB.CLOSE (
   file IN INTEGER);
```

  Table 10-197 DBE_LOB.CLOSE parameters

Parameter	Description
lob/file	BLOB/CLOB/File object whose LOB descriptor is to be closed

- DBE_LOB.MATCH

  Returns the Mth occurrence position of a string in a LOB or BFILE file. **NULL** is returned for an invalid input. The LOB or BFILE file can be greater than 1 GB, up to 32 TB.

  The prototype of the DBE_LOB.MATCH function is as follows:

```
DBE_LOB.MATCH(
   blob_obj    IN BLOB,
   blob_obj2   IN RAW,
   beg_index   IN BIGINT DEFAULT 1,
   occur_index IN BIGINT DEFAULT 1)
RETURN BIGINT;

DBE_LOB.MATCH(
   clob_obj    IN CLOB,
   clob_obj2   IN VARCHAR2,
   beg_index   IN BIGINT DEFAULT 1,
   occur_index IN BIGINT DEFAULT 1)
RETURN BIGINT;

DBE_LOB.MATCH(
   bfile       IN DBE_LOB.BFILE,
   blob_obj2     IN RAW,
   beg_index   IN BIGINT DEFAULT 1,
   occur_index IN BIGINT DEFAULT 1)
RETURN BIGINT;
```


Table 10-198 DBE_LOB.MATCH parameters

Parameter	Description
blob_obj/ clob_obj/ bfile	BLOB/CLOB descriptor to be searched for, or BFILE (which must be opened using **DBE_LOB.BFILEOPEN** first). If this parameter is left blank, null is returned.
blob_obj 2/ clob_obj2	Pattern to match. It is RAW for BLOB/BFILE objects and VARCHAR2 for CLOBs. If this parameter is left empty, null is returned.
beg_inde x	Absolute offset (in bytes) for BLOB/BFILE objects, or offset (in characters) for CLOBs. The start position for matching is 1. **NOTE** The value ranges from 1 to *LOBMAXSIZE*. If a value out of the range is input, null is returned.
occur_ind ex	Number of pattern matching times. The minimum value is 1. **NOTE** If the value is greater than the maximum number of times that the pattern string can be matched in the LOB, **0** is returned. If the value is out of the range from 1 to *LOBMAXSIZE*, null is returned.

- DBE_LOB.COMPARE

  Compares LOBs or BFILE objects.

  - If the compared objects are equal, **0** is returned. Otherwise, a non-zero value is returned.
  - If the first LOB is smaller than the second, **–1** is returned. If the first LOB is larger than the second, **1** is returned.
  - If any of the *len*, *start1*, and *start2* parameters is invalid, **NULL** is returned. The valid offset range is 1 to *LOBMAXSIZE*.
  - If both the values of **start_pos1** and **start_pos2** exceed the LOB/BFILE length, **0** is returned.

  The prototype of the DBE_LOB.COMPARE function is as follows:

```
DBE_LOB.COMPARE (
   lob1       IN BLOB,
   lob2       IN BLOB,
   len        IN BIGINT DEFAULT 1073741312,
   start_pos1 IN BIGINT DEFAULT 1,
   start_pos2 IN BIGINT DEFAULT 1)
RETURN INTEGER;

DBE_LOB.COMPARE (
   lob1       IN CLOB,
   lob2       IN CLOB,
   len        IN BIGINT DEFAULT 1073741312,
   start_pos1 IN BIGINT DEFAULT 1,
   start_pos2 IN BIGINT DEFAULT 1)
RETURN INTEGER;
DBE_LOB.COMPARE (
   file1      IN DBE_LOB.BFILE,
   file2      IN DBE_LOB.BFILE,
   len        IN BIGINT DEFAULT 1073741312,
   start_pos1 IN BIGINT DEFAULT 1,
   start_pos2 IN BIGINT DEFAULT 1)
RETURN INTEGER;
```


Table 10-199 DBE_LOB.COMPARE parameters

Parameter	Description
lob1/file1	First BLOB/CLOB/BFILE to be compared (which must be opened using **DBE_LOB.BFILEOPEN** first).
lob2/file2	Second BLOB/CLOB/BFILE to be compared (which must be opened using **DBE_LOB.BFILEOPEN** first).
len	Number of characters or bytes to be compared. The default value is **1073741312**.
start_pos1	Offset of the first LOB descriptor. The initial position is 1, and the maximum value is the maximum length of the LOB content.
start_pos2	Offset of the second LOB descriptor. The initial position is 1, and the maximum value is the maximum length of the LOB content.

- DBE_LOB.SUBSTR

  Reads a LOB substring and returns the read substring.

  The prototype of the DBE_LOB.SUBSTR function is as follows:

```
DBE_LOB.SUBSTR(
   lob_loc IN BLOB,
   amount  IN INTEGER DEFAULT 32767,
   off_set IN INTEGER  DEFAULT 1)
RETURN RAW;


DBE_LOB.SUBSTR(
   lob_loc IN CLOB,
   amount  IN INTEGER DEFAULT 32767,
   off_set IN INTEGER  DEFAULT 1)
RETURN VARCHAR2;
```

Table 10-200 DBE_LOB.SUBSTR parameters

Parameter	Description
lob_loc	LOB descriptor whose substring is to be read. For BLOBs, the return value is of the RAW type. For CLOBs, the return value is of the VARCHAR2 type.
amount	Number of bytes or characters to be read. **NOTE** The value ranges from 1 to 32767. If the value exceeds the range, null is returned.
off_set	Number of characters or bytes offset from the start position. **NOTE** The value ranges from 1 to *LOBMAXSIZE*. If the value exceeds the range, null is returned.

- DBE_LOB.LOB_SUBSTR

Reads a LOB or BFILE substring and returns the read substring. The LOB or BFILE file can be greater than 1 GB, up to 32 TB.

The prototype of the DBE_LOB.LOB_SUBSTR function is as follows:

```
DBE_LOB.LOB_SUBSTR(
   lob_loc IN BLOB,
   amount  IN INTEGER DEFAULT 32767,
   off_set IN BIGINT  DEFAULT 1)
RETURN RAW;

DBE_LOB.LOB_SUBSTR(
   lob_loc IN CLOB,
   amount  IN INTEGER DEFAULT 32767,
   off_set IN BIGINT  DEFAULT 1)
RETURN VARCHAR2;

DBE_LOB.LOB_SUBSTR(
   bfile   IN DBE_LOB.BFILE,
   amount  IN INTEGER DEFAULT 32767,
   off_set IN BIGINT  DEFAULT 1)
RETURN RAW;
```

Table 10-201 DBE_LOB.LOB_SUBSTR parameters

Parameter	Description
lob_loc/ bfile	LOB descriptor or BFILE file whose substring is to be read. The file must be opened by DBE__LOB.BFILEOPEN first. For BLOBs/ BFILE files, the return value is of the RAW type. For CLOBs, the return value is of the VARCHAR2 type.
amount	Number of bytes or characters to be read. **NOTE** The value ranges from 1 to 32767. If the value exceeds the range, null is returned.
off_set	Number of characters or bytes offset from the start position. **NOTE** The value ranges from 1 to *LOBMAXSIZE*. If the value exceeds the range, null is returned.

- DBE_LOB.STRIP

  Truncates a LOB based on a specified length. After this stored procedure is executed, the length of the LOB is set to the length specified by the **newlen** parameter.

  The prototype of the DBE_LOB.STRIP function is as follows:

  ```
  DBE_LOB.STRIP(
     lob_loc INOUT BLOB,
     newlen  IN INTEGER);

  DBE_LOB.STRIP(
     lob_loc INOUT CLOB,
     newlen  IN INTEGER);
  ```


Table 10-202 DBE_LOB.STRIP parameters

Parameter	Description
lob_loc	LOB to read as the IN parameter, or truncated object as the OUT parameter. If this parameter is left empty, an error is reported.
newlen	New length after truncation, in bytes for BLOBs or in characters for CLOBs.

- DBE_LOB.LOB_STRIP

  Truncates a LOB based on a specified length. After this stored procedure is executed, the length of the LOB is set to the length specified by the **newlen** parameter. The LOB can be greater than 1 GB, up to 32 TB.

  The prototype of the DBE_LOB.LOB_STRIP function is as follows:

```
DBE_LOB.LOB_STRIP(
   lob_loc INOUT BLOB,
   newlen  IN    BIGINT);

DBE_LOB.LOB_STRIP(
   lob_loc INOUT CLOB,
   newlen  IN    BIGINT);
```

Table 10-203 DBE_LOB.LOB_STRIP parameters

Parameter	Description
lob_loc	LOB to read as the IN parameter, or truncated object as the OUT parameter. If this parameter is left empty, an error is reported.
newlen	New length after truncation, in bytes for BLOBs or in characters for CLOBs. **NOTE** If the value is less than 1, null is returned. If the value is greater than the LOB length, an error is reported.

- DBE_LOB.CREATE_TEMPORARY

  Creates a temporary BLOB or CLOB. This API is used only for syntax compatibility.

  The prototype of the DBE_LOB.CREATE_TEMPORARY function is as follows:

```
DBE_LOB.CREATE_TEMPORARY (
   lob_loc INOUT BLOB,
   cache   IN    BOOLEAN,
   dur     IN    INTEGER DEFAULT 10);

DBE_LOB.CREATE_TEMPORARY (
   lob_loc INOUT CLOB,
   cache   IN    BOOLEAN,
   dur     IN    INTEGER DEFAULT 10);
```


Table 10-204 DBE_LOB.CREATE_TEMPORARY parameters

Parameter	Description
lob_loc	LOB descriptor
cache	Used only for syntax compatibility.
dur	Used only for syntax compatibility.

- DBE_LOB.APPEND

  Appends *source_obj* to the end of the target LOB.

  The prototype of the DBE_LOB.APPEND function is as follows:

```
DBE_LOB.APPEND (
   blob_obj INOUT BLOB,
   source_obj  IN    BLOB);

DBE_LOB.APPEND (
   clob_obj INOUT CLOB,
   source_obj  IN    CLOB);
```

Table 10-205 DBE_LOB.APPEND parameters

Parameter	Description
blob_obj/ clob_obj	BLOB/CLOB to which the buffer content is to be written
source_obj	BLOB/CLOB from which the buffer content is to be read

- DBE_LOB.LOB_APPEND

  Appends *source_obj* to the end of the target LOB.

  The prototype of the DBE_LOB.LOB_APPEND function is as follows:

```
DBE_LOB.LOB_APPEND(
   blob_obj   INOUT BLOB,
   source_obj IN    BLOB);

DBE_LOB.LOB_APPEND(
   clob_obj   INOUT CLOB,
   source_obj IN    CLOB);
```

Table 10-206 DBE_LOB.LOB_APPEND parameters

Parameter	Description
blob_obj/ clob_obj	BLOB/CLOB to which the buffer content is to be written
source_obj	BLOB/CLOB from which the buffer content is to be read

- DBE_LOB.FREETEMPORARY

  Frees LOB files created by CREATE_TEMPORARY.

  The prototype of the DBE_LOB.FREETEMPORARY function is as follows:

```
DBE_LOB.FREETEMPORARY (
   blob INOUT BLOB);
```



```
DBE_LOB.FREETEMPORARY (
   clob INOUT CLOB);
```

Table 10-207 DBE_LOB.FREETEMPORARY parameters

Parameter	Description
blob/clob	BLOB/CLOB to be freed.

- DBE_LOB.FILEOPEN

  Opens an external database BFILE file and returns its file descriptor. A maximum of 10 BFILE files can be opened in a session. The BFILE type is defined as follows:

  ```
  DBE_LOB.BFILE (
     directory TEXT,
     filename  TEXT,
     fd        INTEGER);
  ```

  The prototype of the DBE_LOB.FILEOPEN function is as follows:

  ```
  DBE_LOB.FILEOPEN (
     bfile     IN DBE_LOB.BFILE,
     open_mode IN TEXT)
  RETURN INTEGER;
  ```

  Table 10-208 DBE_LOB.FILEOPEN parameters

Parameter	Description
bfile	External database file to be opened. For a BFILE file, this parameter specifies the file path , file name, and file descriptor. **NOTE** The *file* variable contains the location of the file directory *directory* and the file name *filename*. • Location of the file directory, which needs to be added to system catalog **20.2.57-PG_DIRECTORY**. If the input path does not match the path in **20.2.57-PG_DIRECTORY**, an error indicating that the path does not exist is reported. • When the GUC parameter **safe_data_path** is enabled, you can only use an advanced package to read and write files in the file path specified by **safe_data_path**. • File name with an extension (file type), excluding the path name. The path contained in the file name will be ignored by OPEN. In Unix, the file name cannot end with the combination of a slash and a period (/.).
open_mode	File open mode, which can only be **r** (that is, read). An error is reported in other modes.

- DBE_LOB.FILECLOSE

  Closes an external BFILE file.

  The prototype of the DBE_LOB.FILECLOSE function is as follows:

  ```
  DBE_LOB.FILECLOSE (
     file IN INTEGER);
  ```


Table 10-209 DBE_LOB.FILECLOSE parameters

Parameter	Description
file	External database file to be closed (that is opened by FILEOPEN).

- DBE_LOB.BFILEOPEN

  Opens an external database BFILE file. A maximum of 10 BFILE files can be opened in a session.

  The prototype of DBE_LOB.BFILEOPEN is as follows:

```
DBE_LOB.BFILEOPEN (
   bfile     INOUT DBE_LOB.BFILE,
   open_mode IN    TEXT DEFAULT 'R');
```

Table 10-210 DBE_LOB.BFILEOPEN parameters

Parameter	Description
bfile	Opened database BFILE file as the INOUT parameter. **NOTE** The *bfile* variable contains the location of the file directory *directory* and the file name *filename*. • Location of the file directory, which needs to be added to system catalog **20.2.57-PG_DIRECTORY**. If the input path does not match the path in **20.2.57-PG_DIRECTORY**, an error indicating that the path does not exist is reported. • When the GUC parameter **safe_data_path** is enabled, you can only use an advanced package to read and write files in the file path specified by **safe_data_path**. • File name with an extension (file type), excluding the path name. The path contained in the file name will be ignored by OPEN. In Unix, the file name cannot end with the combination of a slash and a period (/.).
open_mode	File open mode, which can only be **r** (that is, read). An error is reported in other modes.

Examples

```
-- Obtain the substring of the BFILE (the file content is ABCD).
DECLARE
bfile dbe_lob.bfile;
BEGIN
bfile = DBE_LOB.BFILENAME(dir_name, file_name); -- Obtain the corresponding BFILE.
DBE_LOB.bfileopen(bfile, 'r'); -- Open the BFILE.
RAISE NOTICE 'res:%', DBE_LOB.lob_substr(bfile, 1, 1); -- Obtain and print the substring.
DBE_LOB.bfileclose(bfile);-- Close the BFILE.
END;
/
NOTICE:  res:41
ANONYMOUS BLOCK EXECUTE
```

- DBE_LOB.BFILECLOSE

  Closes an external database BFILE file.

  The prototype of DBE_LOB.BFILECLOSE is as follows:


```
DBE_LOB.BFILECLOSE (
   bfile INOUT DBE_LOB.BFILE);
```

Table 10-211 DBE_LOB.BFILECLOSE parameters

Parameter	Description
bfile	Closed database BFILE file as the INOUT parameter.

- DBE_LOB.LOADFROMFILE

  Loads an external BFILE file to a BLOB and returns the object of the RAW type.

  The prototype of the DBE_LOB.LOADFROMFILE function is as follows:

```
DBE_LOB.LOADFROMFILE (
   dest_lob    IN BLOB,
   src_file    IN INTEGER,
   amount      IN INTEGER,
   dest_offset IN INTEGER,
   src_offset  IN INTEGER)
RETURN RAW;
```

Table 10-212 DBE_LOB.LOADFROMFILE parameters

Parameter	Description
dest_lob	Target BLOB. The BFILE file will be loaded to the specified offset position of the BLOB.
src_bfile	Source BFILE file to be read.
amount	Length of the content to be read from the BFILE file. **NOTE** If the length is less than 1 or greater than 32767, an error is reported.
dest_offset	Offset length of the BLOB. **NOTE** If the offset is less than 1 or greater than the value of *LOBMAXSIZE*, an error is reported.
src_offset	Offset length of the BFILE file. **NOTE** • If the offset is less than 1 or greater than the value of *LOBMAXSIZE*, an error is reported. • If the sum of *amount* and *src_offset* is greater than the length of *src_bfile* plus 1, an error is reported.

- DBE_LOB.LOADFROMBFILE

  Loads an external BFILE file to a LOB.

  The prototype of the DBE_LOB.LOADFROMBFILE function is as follows:

```
DBE_LOB.LOADFROMBFILE (
   dest_lob    INOUT BLOB,
   src_file    IN    DBE_LOB.BFILE,
   amount      IN    BIGINT,
   dest_offset IN    BIGINT DEFAULT 1,
```



```
   src_offset  IN    BIGINT DEFAULT 1)
RETURN BLOB;

DBE_LOB.LOADFROMBFILE (
   dest_lob    INOUT CLOB,
   src_file    IN    DBE_LOB.BFILE,
   amount      IN    BIGINT,
   dest_offset IN    BIGINT DEFAULT 1,
   src_offset  IN    BIGINT DEFAULT 1)
RETURN CLOB;
```

Table 10-213 DBE_LOB.LOADFROMBFILE parameters

Parameter	Description
dest_lob	Target LOB as the INOUT parameter, to which the BFILE file will be loaded. The file must be opened by **DBE_LOB.BFILEOPEN** first. The LOB can be greater than 1 GB, up to 32 TB.
src_file	Source BFILE file to be read. The BFILE file can be greater than 1 GB, up to 32 TB.
amount	Length of the content to be read from the BFILE file and to be written to the LOB. **NOTE** If the length is less than 1 or greater than the value of *LOBMAXSIZE*, an error is reported.
dest_offset	Offset length of the target LOB **NOTE** If the offset is less than 1 or greater than the value of *LOBMAXSIZE*, an error is reported.
src_offset	Offset length of the BFILE file. **NOTE** • If the offset is less than 1 or greater than the value of *LOBMAXSIZE*, an error is reported. • If the sum of *amount* and *src_offset* is greater than the length of *src_bfile* plus 1, an error is reported.

- DBE_LOB.LOADBLOBFROMFILE

  Loads an external BFILE file to a BLOB and returns the object of the RAW type.

  The prototype of the DBE_LOB.LOADBLOBFROMFILE function is as follows:

```
DBE_LOB.LOADBLOBFROMFILE (
   dest_lob    IN BLOB,
   src_file    IN INTEGER,
   amount      IN INTEGER,
   dest_offset IN INTEGER,
   src_offset  IN INTEGER)
RETURN RAW;
```


Table 10-214 DBE_LOB.LOADBLOBFROMFILE parameters

Parameter	Description
dest_lob	Target BLOB, to which the BFILE file will be loaded.
src_file	Source BFILE file to be read.
amount	Length of the target BLOB. If the length of a file exceeds this threshold, the file will not be saved to the BLOB. **NOTE** If the length is less than 1 or greater than 32767, an error is reported.
dest_offset	Offset length of the BLOB. **NOTE** If the offset is less than 1 or greater than the value of *LOBMAXSIZE*, an error is reported.
src_offset	Offset length of the BFILE file. **NOTE** • If the offset is less than 1 or greater than the value of *LOBMAXSIZE*, an error is reported. • If the sum of *amount* and *src_offset* is greater than the length of *src_bfile* plus 1, an error is reported.

- DBE_LOB.LOADBLOBFROMBFILE

  Loads an external BFILE file to a BLOB.

  The prototype of the DBE_LOB.LOADBLOBFROMBFILE function is as follows:

```
DBE_LOB.LOADBLOBFROMFILE (
   dest_lob     INOUT BLOB,
   src_file     IN     DEB_LOB.BFILE,
   amount       IN     BIGINT,
   dest_offset INOUT BIGINT,
   src_offset  INOUT BIGINT)
```

Table 10-215 DBE_LOB.LOADBLOBFROMBFILE parameters

Parameter	Description
dest_lob	Target BLOB as the INOUT parameter, to which the BFILE file (which must be opened using **DBE_LOB.BFILEOPEN** first) is loaded. The BFILE file can be greater than 1 GB, up to 32 TB.
src_file	Source BFILE file to be read. The BFILE file can be greater than 1 GB, up to 32 TB.
amount	Length of the target BLOB. If the length of a file exceeds this threshold, the file will not be saved to the BLOB. **NOTE** If the length is less than 1 or greater than the value of *LOBMAXSIZE*, an error is reported.

Parameter	Description
dest_offset	Offset length of the BLOB. **NOTE** If the offset is less than 1 or greater than the value of *LOBMAXSIZE*, an error is reported.
src_offset	Offset length of the BFILE file. **NOTE** • If the offset is less than 1 or greater than the value of *LOBMAXSIZE*, an error is reported. • If the sum of *amount* and *src_offset* is greater than the length of *src_bfile* plus 1, an error is reported.

- DBE_LOB.LOADCLOBFROMFILE

  Loads an external BFILE file to a CLOB and returns the object of the RAW type.

  The prototype of the DBE_LOB.LOADCLOBFROMFILE function is as follows:

```
DBE_LOB.LOADCLOBFROMFILE (
   dest_lob    IN CLOB,
   src_file    IN INTEGER,
   amount      IN INTEGER,
   dest_offset IN INTEGER,
   src_offset  IN INTEGER)
RETURN RAW;
```

  Table 10-216 DBE_LOB.LOADCLOBFROMFILE parameters

Parameter	Description
dest_lob	Target CLOB, to which the BFILE file will be loaded.
src_file	Source BFILE file to be read.
amount	Length of the CLOB. **NOTE** If the length is less than 1 or greater than 32767, an error is reported.
dest_offset	Offset length of the CLOB. **NOTE** If the offset is less than 1 or greater than the value of *LOBMAXSIZE*, an error is reported.
src_offset	Offset length of the BFILE file. **NOTE** • If the offset is less than 1 or greater than the value of *LOBMAXSIZE*, an error is reported. • If the sum of *amount* and *src_offset* is greater than the length of *src_bfile* plus 1, an error is reported.

- DBE_LOB.LOADCLOBFROMBFILE

  Loads an external BFILE file to a CLOB.

  The prototype of the DBE_LOB.LOADCLOBFROMBFILE function is as follows:

```
DBE_LOB.LOADCLOBFROMBFILE (
   dest_lob    INOUT CLOB,
   src_file    IN    DEB_LOB.BFILE,
   amount      IN    BIGINT,
   dest_offset INOUT BIGINT,
   src_offset  INOUT BIGINT)
```

Table 10-217 DBE_LOB.LOADCLOBFROMBFILE parameters

Parameter	Description
dest_lob	Target CLOB as the INOUT parameter, to which the BFILE file (which must be opened using **DBE_LOB.BFILEOPEN** first) is loaded. The BFILE file can be greater than 1 GB, up to 32 TB.
src_file	Source BFILE file to be read. The BFILE file can be greater than 1 GB, up to 32 TB.
amount	Length of the target CLOB. If the length of a file exceeds this threshold, the file will not be saved to the CLOB. **NOTE** If the length is less than 1 or greater than the value of *LOBMAXSIZE*, an error is reported.
dest_offset	Offset length of the CLOB. **NOTE** If the offset is less than 1 or greater than the value of *LOBMAXSIZE*, an error is reported.
src_offset	Offset length of the BFILE file. **NOTE** • If the offset is less than 1 or greater than the value of *LOBMAXSIZE*, an error is reported. • If the sum of *amount* and *src_offset* is greater than the length of *src_bfile* plus 1, an error is reported.

- DBE_LOB.CONVERTTOBLOB

  Converts a CLOB to a BLOB. The CLOB cannot be greater than 1 GB.

  The prototype of the DBE_LOB.CONVERTTOBLOB function is as follows:

```
DBE_LOB.CONVERTTOBLOB(
   dest_blob   IN BLOB,
   src_clob    IN CLOB,
   amount      IN INTEGER DEFAULT 32767,
   dest_offset IN INTEGER DEFAULT 1,
   src_offset  IN INTEGER DEFAULT 1)
RETURN RAW;
```

Table 10-218 DBE_LOB.CONVERTTOBLOB parameters

Parameter	Description
dest_blob	Target BLOB, which is converted from a CLOB.
src_clob	Source CLOB to be read.

Parameter	Description
amount	Length of the target CLOB. If the length of a file exceeds this threshold, the file will not be saved to the BLOB. If the length is less than 1 or greater than the value of *LOBMAXSIZE*, an error is reported.
dest_offset	Offset length of the BLOB. If **dest_offset** is set to **1**, data is loaded from the start position of the file. The rest may be deduced by analogy.
src_offset	Offset length of the CLOB. If **src_offset** is set to **1**, data is read from the start position of the file. The rest may be deduced by analogy.

- DBE_LOB.LOB_CONVERTTOBLOB

  Converts a CLOB to a BLOB. The LOB can be greater than 1 GB.

  The prototype of the DBE_LOB.LOB_CONVERTTOBLOB function is as follows:

```
DBE_LOB.LOB_CONVERTTOBLOB(
   dest_blob   INOUT BLOB,
   src_clob    IN    CLOB,
   amount      IN    BIGINT,
   dest_offset INOUT BIGINT,
   src_offset  INOUT BIGINT)
```

  Table 10-219 DBE_LOB.LOB_CONVERTTOBLOB parameters

Parameter	Description
dest_blob	Target BLOB, which is converted from a CLOB.
src_clob	Source CLOB to be read.
amount	Length of the target CLOB. If the length of a file exceeds this threshold, the file will not be saved to the BLOB. If the length is less than 1 or greater than the value of *LOBMAXSIZE*, an error is reported.
dest_offset	Offset length of the BLOB. If **dest_offset** is set to **1**, data is loaded from the start position of the file. The rest may be deduced by analogy. If the offset is less than 1 or greater than the value of *LOBMAXSIZE*, an error is reported.
src_offset	Offset length of the CLOB. If **src_offset** is set to **1**, data is read from the start position of the file. The rest may be deduced by analogy. If the offset is less than 1 or greater than the value of *LOBMAXSIZE*, an error is reported.

- DBE_LOB.CONVERTTOCLOB

  Converts a BLOB to a CLOB. The BLOB cannot be greater than 1 GB.

  The prototype of the DBE_LOB.CONVERTTOCLOB function is as follows:


```
DBE_LOB.CONVERTTOCLOB(
   dest_clob   IN CLOB,
   src_blob    IN BLOB,
   amount      IN INTEGER DEFAULT 32767,
   dest_offset IN INTEGER DEFAULT 1,
   src_offset  IN INTEGER DEFAULT 1)
RETURN text;
```

Table 10-220 DBE_LOB.CONVERTTOCLOB parameters

Parameter	Description
dest_clob	Target CLOB, which is converted from a BLOB.
src_blob	Source BLOB to be read.
amount	Length of the target BLOB. If the length of a file exceeds this threshold, the file will not be saved to the CLOB.
dest_offset	Offset length of the CLOB. If **dest_offset** is set to **1**, data is loaded from the start position of the file. The rest may be deduced by analogy.
src_offset	Offset length of the BLOB. If **src_offset** is set to **1**, data is read from the start position of the file. The rest may be deduced by analogy.

- DBE_LOB.LOB_CONVERTTOCLOB

  Converts a BLOB to a CLOB. The LOB can be greater than 1 GB.

  The prototype of the DBE_LOB.LOB_CONVERTTOCLOB function is as follows:

```
DBE_LOB.LOB_CONVERTTOCLOB(
   dest_clob    INOUT CLOB,
   src_blob    IN    BLOB,
   amount      IN    BIGINT,
   dest_offset INOUT BIGINT,
   src_offset  INOUT BIGINT)
```

Table 10-221 DBE_LOB.LOB_CONVERTTOCLOB parameters

Parameter	Description
dest_clob	Target CLOB, which is converted from a BLOB.
src_blob	Source BLOB to be read.
amount	Length of the target BLOB. If the length of a file exceeds this threshold, the file will not be saved to the CLOB.
dest_offset	Offset length of the CLOB. If **dest_offset** is set to **1**, data is loaded from the start position of the file. The rest may be deduced by analogy. If the offset is less than 1 or greater than the value of *LOBMAXSIZE*, an error is reported.

Parameter	Description
src_offset	Offset length of the BLOB. If **src_offset** is set to **1**, data is read from the start position of the file. The rest may be deduced by analogy. If the offset is less than 1 or greater than the value of *LOBMAXSIZE*, an error is reported.

- DBE_LOB.GETCHUNKSIZE

  Returns *TOAST_MAX_CHUNK_SIZE*. When LOB data is stored in the database, TOAST is used internally.

  The prototype of the DBE_LOB.GETCHUNKSIZE function is as follows:

```
DBE_LOB.GETCHUNKSIZE(
   lob_loc IN CLOB
)RETURN INTEGER

DBE_LOB.GETCHUNKSIZE(
   lob_loc IN BLOB
)RETURN INTEGER
```

  Table 10-222 DBE_LOB.GETCHUNKSIZE parameters

Parameter	Description
lob_loc	Target CLOB/BLOB.

- DBE_LOB.LOB_WRITE

  Reads the specified length of the source object from the start position, writes the content to the specified offset position of the target LOB, overrides the original content, and returns the target LOB.

  The prototype of the DBE_LOB.LOB_WRITE function is as follows:

```
DBE_LOB.LOB_WRITE(
   clob_obj INOUT CLOB,
   amount   IN    INTEGER,
   off_set  IN    BIGINT,
   source   IN    VARCHAR2
)
RETURN CLOB;

DBE_LOB.LOB_WRITE(
   blob_obj INOUT BLOB,
   amount   IN    INTEGER,
   off_set  IN    BIGINT,
   source   IN    RAW
)
RETURN BLOB;
```


Table 10-223 DBE_LOB.LOB_WRITE parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
blob_obj/clob_obj	BLOB/CLOB	INOUT	No	Target LOB as the INOUT parameter, to which the content is to be written.
amount	INTEGER	IN	No	Length of the data to be written (in bytes for BLOBs or in characters for CLOBs)
off_set	BIGINT	IN	No	Offset position for writing data to *blob_obj/clob_obj*
source	RAW/VARCHAR2	IN	No	Source object

- DBE_LOB.BFILENAME

  Constructs a BFILE based on the directory and file name.

  The prototype of DBE_LOB.BFILENAME is as follows:

```
DBE_LOB.BFILENAME(
    directory IN TEXT,
    filename  IN TEXT)
RETURN DBE_LOB.BFILE;
```

Table 10-224 DBE_LOB.BFILENAME parameters

Parameter	Description
directory	File path **NOTE** Location of the file directory, which needs to be added to system catalog **20.2.57-PG_DIRECTORY**. If the input path does not match the path in **20.2.57-PG_DIRECTORY**, an error indicating that the path does not exist is reported. • When the GUC parameter **safe_data_path** is enabled, you can only use an advanced package to read and write files in the file path specified by **safe_data_path**. • File name with an extension (file type), excluding the path name. The path contained in the file name will be ignored by OPEN. In Unix, the file name cannot end with the combination of a slash and a period (/.).
filename	File name

Examples

```
-- Obtain the length of a string.
SELECT DBE_LOB.GET_LENGTH('12345678');
 get_length
------------
          8
(1 row)

-- DBE_LOB.READ API
DECLARE
myraw  RAW(100);
amount INTEGER :=2;
buffer INTEGER :=1;
begin
DBE_LOB.READ('123456789012345',amount,buffer,myraw);
dbe_output.print_line(myraw);
end;
/
0123
ANONYMOUS BLOCK EXECUTE

CREATE TABLE blob_Table (t1 blob);
CREATE TABLE blob_Table_bak (t2 blob);
INSERT INTO blob_Table VALUES('abcdef');
INSERT INTO blob_Table_bak VALUES('22222');

-- Multiple DBE_LOB APIs
DECLARE
str varchar2(100) := 'abcdef';
source raw(100);
dest blob;
copyto blob;
amount int;
PSV_SQL varchar2(100);
PSV_SQL1 varchar2(100);
a int :=1;
len int;
BEGIN
source := dbe_raw.cast_from_varchar2_to_raw(str);
amount := dbe_raw.get_length(source);

PSV_SQL :='select * from blob_Table for update';
PSV_SQL1 := 'select * from blob_Table_bak for update';

EXECUTE IMMEDIATE PSV_SQL into dest;
EXECUTE IMMEDIATE PSV_SQL1 into copyto;

DBE_LOB.WRITE(dest, amount, 1, source);
DBE_LOB.WRITE_APPEND(dest, amount, source);

DBE_LOB.ERASE(dest, a, 1);
DBE_OUTPUT.PRINT_LINE(a);
DBE_LOB.COPY(copyto, dest, amount, 10, 1);
perform DBE_LOB.CLOSE(dest);
RETURN;
END;
/
1
ANONYMOUS BLOCK EXECUTE

-- Drop the table.
DROP TABLE blob_Table;
DROP TABLE blob_Table_bak;
```


10.12.2.8 DBE_MATCH

API Description

Table 10-225 provides all APIs supported by the **DBE_MATCH** package.

Table 10-225 DBE_MATCH

API	Description
DBE_MATCH.EDIT_DISTANCE_SIMILARITY	Compares the difference between two character strings (minimum steps of deletion, addition, and conversion) and normalizes the difference to a value ranging from 0 to 100. The value **100** indicates that the two character strings are the same, and the value **0** indicates that the two character strings are different.

- DBE_MATCH.EDIT_DISTANCE_SIMILARITY

  Compares the difference between two character strings (minimum steps of deletion, addition, and conversion) and normalizes the difference to a value ranging from 0 to 100. The value **100** indicates that the two character strings are the same, and the value **0** indicates that the two character strings are different. The prototype of the DBE_MATCH.EDIT_DISTANCE_SIMILARITY function is as follows:

```
DBE_MATCH.EDIT_DISTANCE_SIMILARITY(
    str1 IN text,
    str2 IN text
)returns integer ;
```

Table 10-226 DBE_MATCH.EDIT_DISTANCE_SIMILARITY parameters

Parameter	Description
str1	First character string. If the value is **null**, **0** is returned.
str2	Second character string. If the value is **null**, **0** is returned.

10.12.2.9 DBE_OUTPUT

NOTE

When DBE_OUTPUT.PUT_LINE is used to print the result obtained by the DBE_FILE.READ_LINE_NCHAR API, ensure that the UTF-8 character set encoding can be converted to the current database character set encoding. If the preceding conditions are met, the result can be properly output. DBE_OUTPUT.PRINT_LINE does not support this function.

API Description

Table 10-227 provides all APIs supported by the **DBE_OUTPUT** package.

Table 10-227 DBE_OUTPUT

API	Description
DBE_OUTPUT.PRINT_LINE	Outputs the specified text with newline characters.
DBE_OUTPUT.PRINT	Outputs the specified text without newline characters.
DBE_OUTPUT.SET_BUFFER_SIZE	Sets the size of the output buffer. If the size is not specified, the buffer can contain a maximum of 20000 bytes. If the size is set to a value less than or equal to 2000 bytes, the buffer can contain a maximum of 2000 bytes.
DBE_OUTPUT.DISABLE	Disables the calling of PUT, PUT_LINE, NEW_LINE, GET_LINE and GET_LINES, and clears the output buffer.
DBE_OUTPUT.ENABLE	Enables the buffer, allows the calling of PUT, PUT_LINE, NEW_LINE, GET_LINE, and GET_LINES, and sets the buffer size.
DBE_OUTPUT.GET_LINE	Obtains a line of data from the buffer with a newline character as the boundary. The obtained data is not output to the client.
DBE_OUTPUT.GET_LINES	Obtains the character string of a specified number of lines in the buffer as a VARCHAR array. The obtained content is cleared from the buffer and is not output to the client.
DBE_OUTPUT.NEW_LINE	Places a line at the end of the buffer, places an end-of-line marker, and leaves a new line empty.
DBE_OUTPUT.PUT	Places an input string in the buffer without a newline character at the end. After the anonymous block is executed, the line ending with the newline character is displayed.
DBE_OUTPUT.PUT_LINE	Places an input string in the buffer with a newline character at the end. After the anonymous block is executed, the line ending with the newline character is displayed.

- DBE_OUTPUT.PRINT_LINE

  Outputs the specified text and adds a newline character. The function prototype of **DBE_OUTPUT.PRINT_LINE** is as follows:

  ```
  DBE_OUTPUT.PRINT_LINE (
  format IN VARCHAR2);
  ```

  Table 10-228 DBE_OUTPUT.PRINT_LINE parameters

Parameter	Description
format	Output text.

- DBE_OUTPUT.PRINT

  Outputs the specified text without adding a newline character. The function prototype of **DBE_OUTPUT.PRINT** is as follows:

  ```
  DBE_OUTPUT.PRINT (
  format IN VARCHAR2);
  ```

  Table 10-229 DBE_OUTPUT.PRINT parameters

Parameter	Description
format	Output text.

- DBE_OUTPUT.SET_BUFFER_SIZE

  The stored procedure **SET_BUFFER_SIZE** sets the output buffer size. If the size is not specified, it contains a maximum of 20000 bytes. The function prototype of **DBE_OUTPUT.SET_BUFFER_SIZE** is as follows:

  ```
  DBE_OUTPUT.SET_BUFFER_SIZE (
  size IN INTEGER default 20000);
  ```

  Table 10-230 DBE_OUTPUT.SET_BUFFER_SIZE parameters

Parameter	Description
size	Sets the output buffer size.

- DBE_OUTPUT.DISABLE

  Disables the calling of PUT, PUT_LINE, NEW_LINE, GET_LINE, and GET_LINES, and clears the output buffer. The prototype of the DBE_OUTPUT.DISABLE function is as follows:

  ```
  DBE_OUTPUT.DISABLE;
  ```

- DBE_OUTPUT.ENABLE

  Enables the buffer, allows the calling of PUT, PUT_LINE, NEW_LINE, GET_LINE, and GET_LINES, and sets the buffer size. If the buffer size is not specified, 20,000 bytes are allowed by default. The prototype of the DBE_OUTPUT.ENABLE function is as follows:


```
DBE_OUTPUT.ENABLE (
   buffer_size IN INTEGER DEFAULT 20000
);
```

Table 10-231 DBE_OUTPUT.ENABLE parameters

Parameter	Description
buffer_size	Upper limit of the buffer size, in bytes. If **buffer_size** is set to **NULL**, the default value (20000 bytes) is used.

- DBE_OUTPUT.GET_LINE

  The stored procedure **GET_LINE** obtains a line of data from the buffer with a newline character as the boundary. The obtained data is not output to the client. The prototype of the DBE_OUTPUT.GET_LINE function is as follows:

  ```
  DBE_OUTPUT.GET_LINE (
     line   OUT VARCHAR2,
     status OUT INTEGER
  );
  ```

  Table 10-232 DBE_OUTPUT.GET_LINE parameters

Parameter	Description
line	Indicates the obtained character string.
status	Specifies whether the calling is normal. If a character string is obtained, the value is **0**. Otherwise, the value is **1**.

- DBE_OUTPUT.GET_LINES

  The stored procedure **GET_LINES** obtains the character string of a specified number of lines in the buffer as a VARCHAR array. The obtained content is cleared from the buffer and is not output to the client. The prototype of the DBE_OUTPUT.GET_LINES function is as follows:

  ```
  DBE_OUTPUT.GET_LINES (
     lines    OUT   VARCHAR[],
     numlines IN OUT INTEGER
  );
  ```

  Table 10-233 DBE_OUTPUT.GET_LINES parameters

Parameter	Description
lines	Outputs an array of multi-line strings read from the buffer.
numlines	Inputs the number of lines to be retrieved from the buffer. The output value is the number of lines actually retrieved. If the output value is less than the input value, there are no more lines in the buffer.

- DBE_OUTPUT.NEW_LINE

  The stored procedure **NEW_LINE** places a line at the end of the buffer, places an end-of-line marker, and leaves a new line empty. The prototype of the DBE_OUTPUT.NEW_LINE function is as follows:


```
DBE_OUTPUT.NEW_LINE;
```

- DBE_OUTPUT.PUT

  The stored procedure **PUT** places an input string in the buffer without a newline character at the end. When the stored procedure ends, the line ending with the newline character is displayed. The prototype of the DBE_OUTPUT.PUT function is as follows:

  ```
  DBE_OUTPUT.PUT (
     item IN VARCHAR2);
  ```

  Table 10-234 DBE_OUTPUT.PUT parameters

Parameter	Description
item	Character or character string to be placed in the buffer.

- DBE_OUTPUT.PUT_LINE

  The stored procedure **PUT_LINE** places an input string in the buffer with a newline character at the end. When the stored procedure ends, the line ending with the newline character is displayed. The prototype of the DBE_OUTPUT.PUT_LINE function is as follows:

  ```
  DBE_OUTPUT.PUT_LINE (
     item IN VARCHAR2);
  ```

  Table 10-235 DBE_OUTPUT.PUT_LINE parameters

Parameter	Description
item	Character or character string to be placed in the buffer.

Examples

```
BEGIN
    DBE_OUTPUT.SET_BUFFER_SIZE(50);
    DBE_OUTPUT.PRINT('hello, ');
    DBE_OUTPUT.PRINT_LINE('database!');-- Output "hello, database!"
END;
/
-- The expected result is as follows:
hello, database!
ANONYMOUS BLOCK EXECUTE

-- Test DISABLE: Disable the calling of PUT, PUT_LINE, NEW_LINE, GET_LINE, and GET_LINES. The PUT_LINE has no output.
BEGIN
 dbe_output.disable();
        dbe_output.put_line('1');
END;
/
-- The expected result is as follows:
ANONYMOUS BLOCK EXECUTE

-- Test ENABLE: Enable the calling of PUT, PUT_LINE, NEW_LINE, GET_LINE, and GET_LINES. The output of PUT_LINE is 1.
BEGIN
 dbe_output.enable();
        dbe_output.put_line('1');
END;
/
```



```
-- The expected result is as follows:
1
ANONYMOUS BLOCK EXECUTE

-- Test PUT: Place the input character string a in the buffer without adding a newline character at the end.
a has no output.
BEGIN
 dbe_output.enable();
 dbe_output.put('a');
END;
/
-- The expected result is as follows:
ANONYMOUS BLOCK EXECUTE

-- Test NEW_LINE. Add a new line. The output is a.
BEGIN
 dbe_output.enable();
 dbe_output.put('a');
        dbe_output.new_line;
END;
/
-- The expected result is as follows:
a
ANONYMOUS BLOCK EXECUTE

-- Test GET_LINE: Obtain buffer data and save the data to variables and use PUT_LINE to output the data.
DECLARE
 line VARCHAR(32672);
 status INTEGER := 0;
BEGIN
 dbe_output.put_line('hello');
 dbe_output.get_line(line, status);
 dbe_output.put_line('-----------');
 dbe_output.put_line(line);
 dbe_output.put_line(status);
END;
/
-- The expected result is as follows:
-----------
hello
0
ANONYMOUS BLOCK EXECUTE

-- Test get_lines: Obtain multiple lines of content in the buffer and use PUT_LINE to output the content.
DECLARE
    lines varchar[];
    linenum integer;
BEGIN
    dbe_output.put_line('line 1');
    dbe_output.put_line('line 2');
    dbe_output.put_line('line 3');
    linenum := 100;
    dbe_output.get_lines(lines, linenum);
    dbe_output.put_line('num: ' || linenum);
    FOR i IN 1 .. linenum LOOP
        dbe_output.put_line(lines[i]);
    END LOOP;
END;
/
-- The expected result is as follows:
num: 3
line 1
line 2
line 3
ANONYMOUS BLOCK EXECUTE
```


NOTE

When the character encoding type server_encoding of the server is not UTF-8, if the character encoding in the data is valid UTF-8 encoding, when processing the character encoding, the DBE_OUTPUT.PUTLINE and DBE_OUTPUT.PUT functions convert the character encoding into the server_encoding encoding format based on the UTF8 encoding logic before performing subsequent operations, as a result, the return values of the functions may not meet the expectation or an error may be reported. After the GUC parameter **enable_convert_illegal_char** is enabled, the DBE_OUTPUT.PUTLINE and DBE_OUTPUT.PUT functions process such codes in the same way as when the parameter is disabled. Special characters are not output as placeholders, therefore, you are advised not to use the DBE_OUTPUT.PUTLINE and DBE_OUTPUT.PUT functions for data containing special characters.

10.12.2.10 DBE_PROFILER

The plprofiler provides an API for analyzing applications of stored procedures. It can collect, store, and delete data related to stored procedures.

Overview

Finding performance problems in PL/SQL functions and stored procedures may be difficult. The only query visible in the system or extended view is sent from the client. This is just the outermost stored procedure call. The plprofiler extension can be used to quickly identify the most time-consuming stored procedures and view the time consumption of a single statement.

You can use this tool to create a stored procedure and related table information in a session, call the plprofiler API to profiling the stored procedure, and then execute the stored procedure. In this case, the analysis data of the stored procedure is generated. You can query the system catalog to obtain data. **Table 5 DBE_PROFILER.PL_PROFILING_FUNCTIONS** lists the stored procedures involved in the profiling. **Table 6 DBE_PROFILER.PL_PROFILING_DETAILS** records details about the execution time of each statement in a stored procedure. **Table 7 DBE_PROFILER.PL_PROFILING_CALLGRAPH** records call stack information and the overall execution time of each stored procedure. **Table 8 DBE_PROFILER.PL_PROFILING_TRACKINFO** records the execution time of each phase of the stored procedure. You can call DBE_PROFILER.PL_CLEAR_PROFILING to delete one piece of data or all data stored in the system catalog after profiling.

After pl_start_profiling is executed, the dbe_profiler tool initializes the memory to prepare for data recording, stubs the stored procedures that are executed subsequently, records detailed information at key points, and records the information in the memory. Data in the memory is written to the four system catalogs only after the transaction is committed. When using the dbe_profiler tool in a transaction, note that data in the memory is not written to the system catalog because the transaction is not committed during transaction execution. Data is written to the system catalog only after the commit operation is executed. Similarly, after the rollback operation is executed, all data is deleted.

Permissions

Common users can access the DBE_PROFILER.PL_PROFILING_FUNCTIONS, DBE_PROFILER.PL_PROFILING_DETAILS, DBE_PROFILER.PL_PROFILING_CALLGRAPH and

DBE_PROFILER.PL_PROFILING_TRACKINFO catalogs and have only the permission to query records.

APIs and System Catalogs

Table 1 DBE_PROFILER lists all APIs supported by the DBE_PROFILER advanced package.

Table 10-236 DBE_PROFILER

API	Description
DBE_PROFILER.PL_START_PLPROFILING	Allows users to perform profiling on target stored procedures.
DBE_PROFILER.PL_CLEAR_PROFILING	Deletes one piece of data or all data stored in system catalogs after profiling.

- DBE_PROFILER.PL_START_PLPROFILING

  Allows users to perform profiling on target stored procedures. The function prototype of the DBE_PROFILER.PL_START_PLPROFILING function is as follows:

  ```
  DBE_PROFILER.PL_START_PLPROFILING (run_id varchar2(64)) return void;
  ```

  Table 10-237 DBE_PROFILER.PL_START_PLPROFILING parameters

Parameter	Description
run_id	Execution ID of profiling. The system combines the session ID and execution ID into a character string in the format of sessionid_runid and writes the character string to the table as the unique ID of profiling.

- DBE_PROFILER.PL_CLEAR_PROFILING

  Deletes one piece of data or all data stored in system catalogs after profiling. The function prototype of the DBE_PROFILER.PL_CLEAR_PROFILING function is as follows:

  ```
  DBE_PROFILER.PL_CLEAR_PROFILING (run_id varchar2) return void;
  ```

  Table 10-238 DBE_PROFILER.PL_CLEAR_PROFILING parameters

Parameter	Description
run_id	If this parameter is not left empty, the unique ID of a profiling data record is deleted. If this parameter is left empty, all profiling data records are deleted.

The analysis data generated by the stored procedure is stored in the database system catalog. The catalog information is as follows:

Table 10-239 Catalog overview

Name
Table 5 DBE_PROFILER.PL_PROFILING_FUNCTIONS
Table 6 DBE_PROFILER.PL_PROFILING_DETAILS
Table 7 DBE_PROFILER.PL_PROFILING_CALLGRAPH
Table 8 DBE_PROFILER.PL_PROFILING_TRACKINFO

- Information about the DBE_PROFILER.PL_PROFILING_FUNCTIONS catalog is as follows:

  Table 10-240 DBE_PROFILER.PL_PROFILING_FUNCTIONS

Column	Type	Description
run_id	VARCHAR2(80) NOT NULL	Unique ID consisting of session ID and execution ID.
funcoid	BIGINT NOT NULL	Stored procedure ID.
schema	TEXT NOT NULL	Schema name.
funcname	TEXT NOT NULL	Stored procedure name.
total_occur	BIGINT	Total number of stored procedure execution times.
total_time	BIGINT	Total execution time of a stored procedure.

- Information about the DBE_PROFILER.PL_PROFILING_DETAILS catalog is as follows:

  Table 10-241 DBE_PROFILER.PL_PROFILING_DETAILS

Column	Type	Description
run_id	VARCHAR2(80) NOT NULL	Unique ID consisting of session ID and execution ID.
funcoid	BIGINT NOT NULL	Stored procedure ID.
line#	INTEGER NOT NULL	Line number.
source	TEXT	Stored procedure execution statement.

Column	Type	Description
cmd_type	TEXT	Statement type.
total_occur	BIGINT	Number of statement execution times.
total_time	BIGINT	Total execution time of a statement.
min_time	BIGINT	Minimum execution time of a statement.
max_time	BIGINT	Maximum execution time of a statement.

- Information about the DBE_PROFILER.PL_PROFILING_CALLGRAPH catalog is as follows:

  Table 10-242 DBE_PROFILER.PL_PROFILING_CALLGRAPH

Column	Type	Description
run_id	varchar2(80) NOT NULL	Unique ID consisting of session ID and execution ID.
stack	text[] NOT NULL	Call stack.
self_time	bigint	Stored procedure execution time.

- Information about the DBE_PROFILER.PL_PROFILING_TRACKINFO catalog is as follows:

  Table 10-243 DBE_PROFILER.PL_PROFILING_TRACKINFO

Column	Type	Description
run_id	VARCHAR2(80) NOT NULL	Unique ID consisting of session ID and execution ID.
funcoid	BIGINT NOT NULL	Stored procedure ID.
step_name	TEXT NOT NULL	Execution phase name.
loops_count	INTEGER	Total number of execution times of the execution phase.
max_time	BIGINT	Maximum execution time of the execution phase.

Column	Type	Description
min_time	BIGINT	Minimum execution time of the execution phase.
avg_time	BIGINT	Average execution time of the execution phase.
total_time	BIGINT	Total execution time of the execution phase.

Constraints

- The dbe_profiler.pl_start_profiling cannot be used in a stored procedure. Calling it in a stored procedure does not take effect.
- The dbe_profiler.pl_start_profiling does not support the function of recording transaction failure statistics to the system catalog.
- The length of the input parameter **runid** of dbe_profiler.pl_start_profiling is limited to 64 characters. If the length exceeds 64 characters, an error is reported.
- If the input parameter of dbe_profiler.pl_start_profiling is the same as the value of **runid** in the profiler system catalog, an error is reported.
- The plprofiler does not support anonymous block statements, PACKAGE initialization statements, and statements in nested subprograms.
- Committing and rolling back an autonomous transaction does not affect the data that has been committed by the primary transaction. Therefore, the plprofiler cannot record information about stored procedures that contain autonomous transactions.

Examples

```
-- Step 1 Create tables and stored procedures.
gaussdb=# DROP TABLE IF EXISTS t1;
gaussdb=# CREATE TABLE t1 (i int);
gaussdb=# CREATE OR REPLACE PROCEDURE p1() AS
    sql_stmt varchar2(200);
    result number;
BEGIN
    for i in 1..1000 loop
      insert into t1 values(1);
    end loop;
    sql_stmt := 'select count(*) from t1';
    EXECUTE IMMEDIATE sql_stmt into result;
END;
/
gaussdb=# CREATE OR REPLACE PROCEDURE p2() AS
BEGIN
   p1();
END;
/
gaussdb=# CREATE OR REPLACE PROCEDURE p3() AS
BEGIN
    p2();
END;
/
```



```
-- Step 2 Call the plprofiler API to profile the stored procedures.
gaussdb=# SELECT dbe_profiler.pl_start_profiling('123');
gaussdb=# CALL p3();

-- Step 3 Query profiling information.
-- Query the dbe_profiler.pl_profiling_functions catalog to check the stored procedures involved in profiling.
gaussdb=# SELECT * FROM dbe_profiler.pl_profiling_functions ORDER BY run_id, funcoid;
       run_id        | funcoid | schema | funcname | total_occur | total_time
---------------------+---------+--------+----------+-------------+------------
 140300887521024_123 |   16770 | public | p1()     |           1 |      54217
 140300887521024_123 |   16771 | public | p2()     |           1 |      54941
 140300887521024_123 |   16772 | public | p3()     |           1 |      55758
(3 rows)
-- Query the dbe_profiler.pl_profiling_details catalog to check the execution time details of each statement
in a stored procedure.
gaussdb=# SELECT * FROM dbe_profiler.pl_profiling_details WHERE funcoid = 16770 ORDER BY run_id,
funcoid, line#;
       run_id        | funcoid | line# |                  source                   |  cmd_type  | total_occur | total_time |
max_time | min_time
---------------------+---------+-------+-------------------------------------------+------------+-------------
+------------+----------+----------
 140300887521024_123 |   16770 |     1 |  DECLARE                                  |            |           0 |          0
|        0 |        0
 140300887521024_123 |   16770 |     2 | sql_stmt varchar2(200);                   |            |           0 |          0
|        0 |        0
 140300887521024_123 |   16770 |     3 | result number;                            |            |           0 |          0
|        0 |        0
 140300887521024_123 |   16770 |     4 | begin                                     |            |           0 |          0 |
0 |        0
 140300887521024_123 |   16770 |     5 |     for i in 1..1000 loop                 | FORI       |           1 |      52496
|    52496 |    52496
 140300887521024_123 |   16770 |     6 |       insert into t1 values(1);           | EXECSQL    |        1000 |
51970 |     2115 |       47
 140300887521024_123 |   16770 |     7 |     end loop;                             |            |           0 |          0 |
0 |        0
 140300887521024_123 |   16770 |     8 |     sql_stmt := 'select count(*) from t1';  | ASSIGN     |           1
|        446 |      446 |      446
 140300887521024_123 |   16770 |     9 |     EXECUTE IMMEDIATE sql_stmt into result; | DYNEXECUTE
|           1 |       1271 |     1271 |     1271
 140300887521024_123 |   16770 |    10 | end                                       |            |           0 |          0 |
0 |        0
(10 rows)
-- Query the dbe_profiler.pl_profiling_callgraph catalog to check the call stack information and the overall
execution time of each stored procedure (total_time and self_time correspond to the total execution time
of the stored procedure at the top of the call stack and the execution time of the stored procedure).
gaussdb=# SELECT * FROM dbe_profiler.pl_profiling_callgraph ORDER BY run_id, stack;
       run_id        |                                  stack                                  | self_time
---------------------+-------------------------------------------------------------------------+-----------
 140300887521024_123 | {"public.p3() oid=16772"}                                               |       817
 140300887521024_123 | {"public.p3() oid=16772","public.p2() oid=16771"}                       |       724
 140300887521024_123 | {"public.p3() oid=16772","public.p2() oid=16771","public.p1() oid=16770"} |
54217
(3 rows)
-- Query the dbe_profiler.pl_profiling_trackinfo catalog to check the execution time of each phase of the
stored procedure.
gaussdb=# SELECT step_name, loops_count FROM dbe_profiler.pl_profiling_trackinfo WHERE funcoid=16770;
  step_name   | loops_count
--------------+-------------
 init         |           1
 package      |           1
 spictx       |           1
 compile      |           1
 exec_context |           1
 execute      |           1
 exec_cursor  |           1
 cleanup      |           1
 finsh        |           1
(9 rows)
```



```
-- Step 4 Delete data from the system catalog.
gaussdb=# SELECT dbe_profiler.pl_clear_profiling('');
gaussdb=# SELECT step_name, loops_count FROM dbe_profiler.pl_profiling_trackinfo WHERE funcoid=16770;
 step_name | loops_count
-----------+-------------
(0 rows)
gaussdb=# DROP TABLE t1;

-- Step 5 Profile the stored procedure that contains an autonomous transaction.
-- Create a table.
gaussdb=# CREATE TABLE t2(a int, b int);
-- Create a stored procedure that contains an autonomous transaction.
gaussdb=# CREATE OR REPLACE PROCEDURE autonomous(a int, b int)  AS
DECLARE
    num3 int := a;
    num4 int := b;
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    insert into t2 values(num3, num4);
    dbe_output.print_line('just use call.');
END;
/
-- Create a common stored procedure that calls an autonomous transaction stored procedure.
gaussdb=# CREATE OR REPLACE PROCEDURE autonomous_1(a int, b int)  AS
DECLARE
BEGIN
    dbe_output.print_line('just no use call.');
    insert into t2 values(666, 666);
    autonomous(a,b);
END;
/
gaussdb=# SELECT dbe_profiler.pl_start_profiling ('100');
gaussdb=# CALL autonomous(11,22);
-- Query table information.
gaussdb=# SELECT * FROM dbe_profiler.pl_profiling_functions ORDER BY run_id, funcoid;
 run_id | funcoid | schema | funcname | total_occur | total_time
--------+---------+--------+----------+-------------+------------
(0 rows)

gaussdb=# SELECT * FROM dbe_profiler.pl_profiling_details ORDER BY run_id, funcoid, line#;
 run_id | funcoid | line# | source | cmd_type | total_occur | total_time | max_time | min_time
--------+---------+-------+--------+----------+-------------+------------+----------+----------
(0 rows)

gaussdb=# SELECT * FROM dbe_profiler.pl_profiling_callgraph ORDER BY run_id, stack;
 run_id | stack | self_time
--------+-------+-----------
(0 rows)

gaussdb=# SELECT * FROM dbe_profiler.pl_profiling_trackinfo ORDER BY run_id, funcoid;
 run_id | funcoid | step_name | loops_count | max_time | min_time | avg_time | total_time
--------+---------+-----------+-------------+----------+----------+----------+------------
(0 rows)

gaussdb=# SELECT dbe_profiler.pl_start_profiling ('101');
gaussdb=# CALL autonomous_1(11,22);
gaussdb=# SELECT * FROM dbe_profiler.pl_profiling_functions ORDER BY run_id, funcoid;
       run_id        | funcoid |   schema   |    funcname    | total_occur | total_time
---------------------+---------+------------+----------------+-------------+------------
 140421237831424_101 |   10422 | dbe_output | print_line()   |           1 |        758
 140421237831424_101 |   16771 | public     | autonomous_1() |           1 |      23855
(2 rows)

gaussdb=# SELECT * FROM dbe_profiler.pl_profiling_details ORDER BY run_id, funcoid, line#;
       run_id        | funcoid | line# |                    source                    | cmd_type | total_occur | total_time |
max_time | min_time
---------------------+---------+-------+----------------------------------------------+----------+-------------+------------
+----------+----------
 140421237831424_101 |   10422 |     1 |                                              |          |           0 |          0 |        0
```



```
|       0
 140421237831424_101 |   10422 |     2 | BEGIN                                  |         |          0 |          0 |       0
|       0
 140421237831424_101 |   10422 |     3 |     PKG_UTIL.io_print(format, true);       | PERFORM  |          1
|       754 |      754 |      754
 140421237831424_101 |   10422 |     4 | END;                                   |         |          0 |          0 |       0
|       0
 140421237831424_101 |   10422 |     5 |                                        |         |          0 |          0 |       0
|       0
 140421237831424_101 |   16771 |     1 |                                        |         |          0 |          0 |       0
|       0
 140421237831424_101 |   16771 |     2 | DECLARE                                |         |          0 |          0 |
0 |       0
 140421237831424_101 |   16771 |     3 | BEGIN                                  |         |          0 |          0 |       0
|       0
 140421237831424_101 |   16771 |     4 | dbe_output.print_line('just no use call.'); | PERFORM  |          1
|       2435 |     2435 |     2435
 140421237831424_101 |   16771 |     5 | insert into t2 values(666, 666);           | EXECSQL  |          1 |
602 |      602 |      602
 140421237831424_101 |   16771 |     6 | autonomous(a,b);                           | PERFORM  |          1 |
20813 |    20813 |    20813
 140421237831424_101 |   16771 |     7 | END                                    |         |          0 |          0 |       0
|       0
(12 rows)

gaussdb=# SELECT * FROM dbe_profiler.pl_profiling_callgraph ORDER BY run_id, stack;
      run_id        |                                 stack                                  | self_time
--------------------+------------------------------------------------------------------------+-----------
 140421237831424_101 | {"public.autonomous_1() oid=16771"}                                    |     23097
 140421237831424_101 | {"public.autonomous_1() oid=16771","dbe_output.print_line() oid=10422"} |
758
(2 rows)

gaussdb=# SELECT * FROM dbe_profiler.pl_profiling_trackinfo ORDER BY run_id, funcoid;
      run_id        | funcoid |  step_name   | loops_count | max_time | min_time | avg_time | total_time
--------------------+---------+--------------+-------------+----------+----------+----------+------------
 140421237831424_101 |   10422 | init         |           1 |        0 |        0 |        0 |          0
 140421237831424_101 |   10422 | package      |           1 |        9 |        9 |        9 |          9
 140421237831424_101 |   10422 | spictx       |           1 |        1 |        1 |        1 |          1
 140421237831424_101 |   10422 | compile      |           1 |      383 |      383 |      383 |        383
 140421237831424_101 |   10422 | exec_context |           1 |        1 |        1 |        1 |          1
 140421237831424_101 |   10422 | execute      |           1 |     1301 |     1301 |     1301 |       1301
 140421237831424_101 |   10422 | exec_cursor  |           1 |        3 |        3 |        3 |          3
 140421237831424_101 |   10422 | cleanup      |           1 |       11 |       11 |       11 |         11
 140421237831424_101 |   10422 | finsh        |           1 |        0 |        0 |        0 |          0
 140421237831424_101 |   16771 | init         |           1 |        0 |        0 |        0 |          0
 140421237831424_101 |   16771 | package      |           1 |      103 |      103 |      103 |        103
 140421237831424_101 |   16771 | spictx       |           1 |        1 |        1 |        1 |          1
 140421237831424_101 |   16771 | compile      |           1 |     1869 |     1869 |     1869 |       1869
 140421237831424_101 |   16771 | exec_context |           1 |        3 |        3 |        3 |          3
 140421237831424_101 |   16771 | execute      |           1 |    24011 |    24011 |    24011 |      24011
 140421237831424_101 |   16771 | exec_cursor  |           1 |        1 |        1 |        1 |          1
 140421237831424_101 |   16771 | cleanup      |           1 |       16 |       16 |       16 |         16
 140421237831424_101 |   16771 | finsh        |           1 |        1 |        1 |        1 |          1
(18 rows)

gaussdb=# SELECT dbe_profiler.pl_clear_profiling('');
gaussdb=# SELECT * FROM dbe_profiler.pl_profiling_functions;
 run_id | funcoid | schema | funcname | total_occur | total_time
--------+---------+--------+----------+-------------+------------
(0 rows)

gaussdb=# DROP TABLE t2;
```


10.12.2.11 DBE_RANDOM

Interface Description

Table 10-244 lists all interfaces supported by the **DBE_RANDOM** package.

Table 10-244 DBE_RANDOM interface parameters

Interface	Description
DBE_RANDOM.SET_SEED	Sets a seed for a random number.
DBE_RANDOM.GET_VALUE	Generates a random number between a specified **low** and a specified **high**.

- DBE_RANDOM.SET_SEED

  The stored procedure **SEED** is used to set a seed for a random number. The prototype of the **DBE_RANDOM.SET_SEED** function is as follows:

  ```
  DBE_RANDOM.SET_SEED (seed  IN  INTEGER);
  ```

  Table 10-245 DBE_RANDOM.SET_SEED interface parameters

Parameter	Description
seed	Generates a seed for a random number.

- DBE_RANDOM.GET_VALUE

  The GET_VALUE function generates a random number between a specified **low** and a specified **high**. The prototype of the **DBE_RANDOM.GET_VALUE** function is as follows:

  ```
  DBE_RANDOM.GET_VALUE(
  min  IN  NUMBER default 0,
  max  IN  NUMBER default 1)
  RETURN NUMBER;
  ```

  Table 10-246 DBE_RANDOM.GET_VALUE interface parameters

Parameter	Description
min	Sets the low bound for a random number. The generated random number is greater than or equal to **min**.
max	Sets the high bound for a random number. The generated random number is less than **max**.

NOTE

- The only requirement is that the parameter type is numeric regardless of the right and left bound values.
- **DBE_RANDOM** implements pseudo-random numbers. Therefore, if the initial value (seed) remains unchanged, the sequence of the pseudo-random numbers also remains unchanged.
- The generated random number contains 15 valid digits.

Examples

```
-- Generate a random number between 0 and 1.
SELECT DBE_RANDOM.GET_VALUE(0,1);
    get_value
------------------
 .917468812743886
(1 row)

-- For integers within a specified range, add the arguments min and max, and truncate the decimals from the result (the maximum value is not included as a possible value). Therefore, for integers from 0 to 99, you can use the following code:
SELECT TRUNC(DBE_RANDOM.GET_VALUE(0,100));
 trunc
-------
    26
(1 row)
```

10.12.2.12 DBE_RAW

The DBE_RAW package provides interfaces for operating RAW types.

NOTICE

RAW data is represented as hexadecimal characters externally, and stored as binary characters internally. For example, the representation of RAW data 11001011 is 'CB', that is, the input for type conversion is 'CB'.

Interface Description

Table 10-247 provides all interfaces supported by the **DBE_RAW** package.

Table 10-247 DBE_RAW

Interface	Description
DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW	Converts an INTEGER value to a RAW value.
DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER	Converts a RAW value to an INTEGER value.
DBE_RAW.GET_LENGTH	Returns the length of a RAW value.
DBE_RAW.CAST_FROM_VARCHAR2_TO_RAW	Converts a VARCHAR2 value to a RAW value.

Interface	Description
DBE_RAW.CAST_TO_VARCHAR2	Converts a RAW value to a VARCHAR2 value.
DBE_RAW.BIT_OR	Returns a RAW value after bitwise OR calculation.
DBE_RAW.SUBSTR	Returns the substring of a RAW value.
DBE_RAW.BIT_AND	Returns a RAW value after bitwise AND calculation.
DBE_RAW.BIT_COMPLEMENT	Returns a RAW value after bitwise complement calculation.
DBE_RAW.BIT_XOR	Returns a RAW value after bitwise XOR calculation.
DBE_RAW.CAST_FROM_BINARY_DOUBLE_TO_RAW	Converts a BINARY_DOUBLE value to a RAW value.
DBE_RAW.CAST_FROM_RAW_TO_BINARY_DOUBLE	Converts a RAW value to a BINARY_DOUBLE value.
DBE_RAW.CAST_FROM_RAW_TO_BINARY_FLOAT	Converts a RAW value to a FLOAT4 value.
DBE_RAW.CAST_FROM_BINARY_FLOAT_TO_RAW	Converts a FLOAT4 value to a RAW value.
DBE_RAW.CAST_FROM_RAW_TO_NUMBER	Converts a RAW value to a NUMERIC value.
DBE_RAW.CAST_FROM_NUMBER_TO_RAW	Converts a NUMERIC value to a RAW value.
DBE_RAW.CAST_FROM_RAW_TO_NVARCHAR2	Converts a RAW value to an NVARCHAR2 value.
DBE_RAW.COMPARE	Returns the first different position of two RAW values.
DBE_RAW.CONCAT	Concatenates a maximum of 12 RAW values into a new RAW value and returns the value.
DBE_RAW.CONVERT	Converts a RAW value from the source encoding mode **from_charset** to the target encoding mode **to_charset**.
DBE_RAW.COPIES	Copies a RAW value for *n* times, concatenates the values, and returns the concatenated result.
DBE_RAW.OVERLAY	Overlays one RAW data with another RAW data by specifying the start position and length to be overlaid.
DBE_RAW.REVERSE	Reverses RAW data by byte.

Interface	Description
DBE_RAW.TRANSLATE	Converts or discards a specified byte of a RAW value.
DBE_RAW.TRANSLITERATE	Converts a specified byte of a RAW value.
DBE_RAW.XRANGE	Returns a RAW value containing the succession of one-byte encodings beginning and ending with the specified byte-code.

- DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW

  Converts an INTEGER value to a RAW value.

  The prototype of the DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW function is as follows:

```
DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW(
    value     IN BIGINT,
    endianess IN INTEGER DEFAULT 1)
RETURN RAW;
```

  Table 10-248 DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW parameters

Parameter	Description
value	INTEGER value to be converted. **NOTE** The DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW expects to process values of the INTEGER type. For compatibility purposes, this parameter is defined as the BIGINT type. The BIGINT type has a larger range than the INTEGER type, which does not affect the interface use.
endianess	Endianness. The value can be **1** (BIG_ENDIAN), **2** (LITTLE_ENDIAN), or **3** (MACHINE_ENDIAN). The default value is **1**. If BIG_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using BIG_ENDIAN. If LITTLE_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using LITTLE_ENDIAN.

- DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER

  Converts a RAW value to an INTEGER value.

  The prototype of the DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER function is as follows:

```
DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER(
    value     IN RAW,
    endianess IN INTEGER DEFAULT 1)
RETURN INTEGER;
```


Table 10-249 DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER parameters

Parameter	Description
value	RAW value to be converted.
endianess	Endianness. The value can be **1**, **2**, or **3**. The value **1** indicates BIG_ENDIAN, **2** indicates LITTLE_ENDIAN, and **3** indicates MACHINE_ENDIAN. The default value is **1**. If BIG_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using BIG_ENDIAN. If LITTLE_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using LITTLE_ENDIAN.

- DBE_RAW.GET_LENGTH

  Returns the length of a RAW value.

  The prototype of the DBE_RAW.GET_LENGTH function is as follows:

```
DBE_RAW.GET_LENGTH(
    value IN RAW)
RETURN INTEGER;
```

Table 10-250 DBE_RAW.GET_LENGTH parameters

Parameter	Description
value	RAW value.

- DBE_RAW.CAST_FROM_VARCHAR2_TO_RAW

  Converts a VARCHAR2 value to a RAW value.

  The prototype of the DBE_RAW.CAST_FROM_VARCHAR2_TO_RAW function is as follows:

```
DBE_RAW.CAST_FROM_VARCHAR2_TO_RAW(
    str IN VARCHAR2)
RETURN RAW;
```

Table 10-251 DBE_RAW.CAST_FROM_VARCHAR2_TO_RAW parameters

Parameter	Description
str	VARCHAR2 value to be converted.

- DBE_RAW.CAST_TO_VARCHAR2

  Converts a RAW value to a VARCHAR2 value.

  The prototype of the DBE_RAW.CAST_TO_VARCHAR2 function is as follows:

```
DBE_RAW.CAST_TO_VARCHAR2(
    str IN RAW)
RETURN VARCHAR2;
```


Table 10-252 DBE_RAW.CAST_TO_VARCHAR2 parameters

Parameter	Description
str	RAW value to be converted.

- DBE_RAW.BIT_OR

  Returns the value of two RAW values after bitwise OR calculation.

  The prototype of the DBE_RAW.BIT_OR function is as follows:

```
DBE_RAW.BIT_OR(
   str1 IN TEXT,
   str2 IN TEXT)
RETURN TEXT;
```

Table 10-253 DBE_RAW.BIT_OR parameters

Parameter	Description
str1	First character string of the bitwise OR calculation. **NOTE** Due to legacy reasons, this parameter is defined as the TEXT type. However, the DBE_RAW.BIT_OR interface expects to process RAW values. Defining this parameter as the TEXT type does not affect the processing of RAW values.
str2	Second character string of the bitwise OR calculation. **NOTE** Due to legacy reasons, this parameter is defined as the TEXT type. However, the DBE_RAW.BIT_OR interface expects to process RAW values. Defining this parameter as the TEXT type does not affect the processing of RAW values.

- DBE_RAW.SUBSTR

  Truncates a RAW value based on the start position *bit off_set* and length *amount*.

  The prototype of the DBE_RAW.SUBSTR function is as follows:

```
DBE_RAW.SUBSTR(
   IN lob_loc BLOB,
   IN off_set INTEGER DEFAULT 1,
   IN amount INTEGER DEFAULT 32767)
RETURN RAW;
```

Table 10-254 DBE_RAW.SUBSTR parameters

Parameter	Description
lob_loc	Source RAW value. **NOTE** Due to legacy reasons, this parameter is defined as the BLOB type. However, the DBE_RAW.SUBSTR interface expects to process RAW values. Defining this parameter as the BLOB type does not affect the processing of RAW values.
off_set	Start position of the substring. The default value is **1**.

Parameter	Description
amount	Substring length. The default value is **32767**.

- DBE_RAW.BIT_AND

  Obtains the bitwise AND result of two RAW values.

  The prototype of the DBE_RAW.BIT_AND function is as follows:

```
DBE_RAW.BIT_AND(
   r1 IN RAW,
   r2 IN RAW)
RETURN RAW;
```

  Table 10-255 DBE_RAW.BIT_AND parameters

Parameter	Description
r1	The RAW value for the bitwise AND operation with r2. The maximum length is 32767.
r2	The RAW value for the bitwise AND operation with r1. The maximum length is 32767.

- DBE_RAW.BIT_COMPLEMENT

  Obtains the bitwise complement result of a RAW value.

  The prototype of the DBE_RAW.BIT_COMPLEMENT function is as follows:

```
DBE_RAW.BIT_COMPLEMENT(
   r IN RAW)
RETURN RAW;
```

  Table 10-256 DBE_RAW.BIT_COMPLEMENT parameters

Parameter	Description
r	The RAW value for the bitwise complement operation. The maximum length is 32767.

- DBE_RAW.BIT_XOR

  Obtains the bitwise XOR result of two RAW values.

  The prototype of the DBE_RAW.BIT_XOR function is as follows:

```
DBE_RAW.BIT_XOR(
   r1 IN RAW,
   r2 IN RAW)
RETURN RAW;
```

  Table 10-257 DBE_RAW.BIT_XOR parameters

Parameter	Description
r1	The RAW value for the bitwise XOR operation with r2. The maximum length is 32767.

Parameter	Description
r2	The RAW value for the bitwise XOR operation with r1. The maximum length is 32767.

- DBE_RAW.CAST_FROM_BINARY_DOUBLE_TO_RAW

  Converts a BINARY_DOUBLE value to a RAW value.

  The prototype of the DBE_RAW.CAST_FROM_BINARY_DOUBLE_TO_RAW function is as follows:

```
DBE_RAW.CAST_FROM_BINARY_DOUBLE_TO_RAW (
   n         IN BINARY_DOUBLE,
   endianess IN INTEGER DEFAULT 1)
RETURN RAW;
```

  Table 10-258 DBE_RAW.CAST_FROM_BINARY_DOUBLE_TO_RAW parameters

Parameter	Description
n	BINARY_DOUBLE value to be converted.
endianess	Endianness. The value can be **1**, **2**, or **3**. The value **1** indicates BIG_ENDIAN, **2** indicates LITTLE_ENDIAN, and **3** indicates MACHINE_ENDIAN. The default value is **1**. If BIG_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using BIG_ENDIAN. If LITTLE_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using LITTLE_ENDIAN.

- DBE_RAW.CAST_FROM_RAW_TO_BINARY_DOUBLE

  Converts a RAW value to a BINARY_DOUBLE value.

  The prototype of the DBE_RAW.CAST_FROM_RAW_TO_BINARY_DOUBLE function is as follows:

```
DBE_RAW.CAST_FROM_RAW_TO_BINARY_DOUBLE(
   r         IN RAW,
   endianess IN INTEGER DEFAULT 1)
RETURN BINARY_DOUBLE;
```

  Table 10-259 DBE_RAW.CAST_FROM_RAW_TO_BINARY_DOUBLE parameters

Parameter	Description
r	The RAW value to be converted. The value contains 8 to 32767 characters.

Parameter	Description
endianess	Endianness. The value can be **1**, **2**, or **3**. The value **1** indicates BIG_ENDIAN, **2** indicates LITTLE_ENDIAN, and **3** indicates MACHINE_ENDIAN. The default value is **1**. If BIG_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using BIG_ENDIAN. If LITTLE_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using LITTLE_ENDIAN.

- DBE_RAW.CAST_FROM_RAW_TO_BINARY_FLOAT

  Converts a RAW value to a FLOAT4 value.

  The prototype of the DBE_RAW.CAST_FROM_RAW_TO_BINARY_FLOAT function is as follows:

```
DBE_RAW.CAST_FROM_RAW_TO_BINARY_FLOAT(
   r         IN RAW,
   endianess IN INTEGER DEFAULT 1)
RETURN FLOAT4;
```

Table 10-260 DBE_RAW.CAST_FROM_RAW_TO_BINARY_FLOAT parameters

Parameter	Description
r	The RAW value to be converted. The value contains 4 to 32767 characters.
endianess	Endianness. The value can be **1**, **2**, or **3**. The value **1** indicates BIG_ENDIAN, **2** indicates LITTLE_ENDIAN, and **3** indicates MACHINE_ENDIAN. The default value is **1**. If BIG_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using BIG_ENDIAN. If LITTLE_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using LITTLE_ENDIAN.

- DBE_RAW.CAST_FROM_BINARY_FLOAT_TO_RAW

  Converts a FLOAT4 value to a RAW value.

  The prototype of the DBE_RAW.CAST_FROM_BINARY_FLOAT_TO_RAW function is as follows:

```
DBE_RAW.CAST_FROM_BINARY_FLOAT_TO_RAW(
   n         IN FLOAT4,
   endianess IN INTEGER DEFAULT 1)
RETURN RAW;
```

Table 10-261 DBE_RAW.CAST_FROM_BINARY_FLOAT_TO_RAW parameters

Parameter	Description
n	FLOAT4 value to be converted.

Parameter	Description
endianess	Endianness. The value can be **1**, **2**, or **3**. The value **1** indicates BIG_ENDIAN, **2** indicates LITTLE_ENDIAN, and **3** indicates MACHINE_ENDIAN. The default value is **1**. If BIG_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using BIG_ENDIAN. If LITTLE_ENDIAN is used on the host where the function is executed, the function execution result using MACHINE_ENDIAN is the same as that using LITTLE_ENDIAN.

- DBE_RAW.CAST_FROM_RAW_TO_NUMBER

  Converts a RAW value to a NUMERIC value.

  NOTE

  The bottom-layer implementation of the number data type in database A is different from that in GaussDB, and the raw data type is the hexadecimal representation of the binary stream implemented at the bottom layer. Therefore, the function in database A is different from that in GaussDB. You cannot obtain the same number-type data from GaussDB based on the raw data corresponding to the number-type data in database A. For details about the number-type data in GaussDB, see the example.

  The prototype of the DBE_RAW.CAST_FROM_RAW_TO_NUMBER function is as follows:

```
DBE_RAW.CAST_FROM_RAW_TO_NUMBER(
    r IN RAW)
RETURN NUMERIC;
```

Table 10-262 DBE_RAW.CAST_FROM_RAW_TO_NUMBER parameters

Parameter	Description
r	The RAW value to be converted. The value contains 6 to 32767 characters.

- DBE_RAW.CAST_FROM_NUMBER_TO_RAW

  Converts a NUMERIC value to a RAW value.

  NOTE

  The bottom-layer implementation of the number data type in database A is different from that in GaussDB, and the raw data type is the hexadecimal representation of the binary stream implemented at the bottom layer. Therefore, the function in database A is different from that in GaussDB. You cannot restore the raw data corresponding to the number-type data in database A to the original number type in database A in the GaussDB database. For details about the performance of this function in GaussDB, see the example.

  The prototype of the DBE_RAW.CAST_FROM_NUMBER_TO_RAW function is as follows:

```
DBE_RAW.CAST_FROM_NUMBER_TO_RAW(
    n IN NUMERIC)
RETURN RAW;
```


Table 10-263 DBE_RAW.CAST_FROM_NUMBER_TO_RAW parameters

Parameter	Description
n	NUMERIC value to be converted.

- DBE_RAW.CAST_FROM_RAW_TO_NVARCHAR2

  Converts a RAW value to an NVARCHAR2 value.

  The prototype of the DBE_RAW.CAST_FROM_RAW_TO_NVARCHAR2 function is as follows:

```
DBE_RAW.CAST_FROM_RAW_TO_NVARCHAR2(
   r IN RAW)
RETURN NVARCHAR2;
```

Table 10-264 DBE_RAW.CAST_FROM_RAW_TO_NVARCHAR2 parameters

Parameter	Description
r	The RAW value to be converted. The maximum length is 32767.

- DBE_RAW.COMPARE

  Returns the first different position of two RAW values.

  The prototype of the DBE_RAW.COMPARE function is as follows:

```
DBE_RAW.COMPARE(
   r1  IN RAW,
   r2  IN RAW,
   pad IN RAW DEFAULT NULL)
RETURN INTEGER;
```

Table 10-265 DBE_RAW.COMPARE parameters

Parameter	Description
r1	First data to be compared. The value can be **NULL** or the length is 0. The maximum length is 32767.
r2	Second data to be compared. The value can be **NULL** or the length is 0. The maximum length is 32767.
pad	The first byte of **pad** is used to extend the shorter one of **r1** or **r2**. The maximum length is 32767. If this parameter is set to **NULL**, the length is 0, or the default value is used, the extended value is **0x00**.

- DBE_RAW.CONCAT

  Concatenates a maximum of 12 RAW values into a new RAW value and returns the value. If the length after concatenation exceeds 32767, an error is reported.

  The prototype of the DBE_RAW.CONCAT function is as follows:

```
DBE_RAW.CONCAT(
   r1  IN RAW DEFAULT NULL,
```



```
    r2  IN RAW DEFAULT NULL,
    r3  IN RAW DEFAULT NULL,
    r4  IN RAW DEFAULT NULL,
    r5  IN RAW DEFAULT NULL,
    r6  IN RAW DEFAULT NULL,
    r7  IN RAW DEFAULT NULL,
    r8  IN RAW DEFAULT NULL,
    r9  IN RAW DEFAULT NULL,
    r10 IN RAW DEFAULT NULL,
    r11 IN RAW DEFAULT NULL,
    r12 IN RAW DEFAULT NULL)
RETURN RAW;
```

Table 10-266 DBE_RAW.CONCAT parameters

Parameter	Description
r1...r12	The RAW values to be concatenated.

- DBE_RAW.CONVERT

  Converts a RAW value from the source encoding mode **from_charset** to the target encoding mode **to_charset**.

  If the rule for converting between source and target encoding (for example, GBK and LATIN1) does not exist, the parameter **r** is returned without conversion. See the pg_conversion system catalog for details.

  The prototype of the DBE_RAW.CONVERT function is as follows:

  ```
  DBE_RAW.CONVERT(
    r           IN RAW,
    to_charset   IN VARCHAR2,
    from_charset IN VARCHAR2)
  RETURN RAW;
  ```

  Table 10-267 DBE_RAW.CONVERT parameters

Parameter	Description
r	The RAW value to be converted. The maximum length is 32767.
to_charset	Name of the target encoding character set.
from_charset	Name of the source encoding character set. In this encoding, **r** must be valid.

- DBE_RAW.COPIES

  Copies a RAW value for *n* times, concatenates the values, and returns the concatenated result. If the length after copying exceeds 32767, an error is reported.

  The prototype of the DBE_RAW.COPIES function is as follows:

  ```
  DBE_RAW.COPIES(
    r IN RAW,
    n IN NUMERIC)
  RETURN RAW;
  ```


Table 10-268 DBE_RAW.COPIES parameters

Parameter	Description
r	The RAW values to be copied.
n	Number of replications. The value must be a positive number. If the value is a decimal, the value is rounded off.

- DBE_RAW.OVERLAY

  Overlays one RAW data with another RAW data by specifying the start position and length to be overlaid.

  The prototype of the DBE_RAW.OVERLAY function is as follows:

```
DBE_RAW.OVERLAY(
   overlay_str IN RAW,
   target      IN RAW,
   pos         IN BINARY_INTEGER DEFAULT 1,
   len         IN BINARY_INTEGER DEFAULT NULL,
   pad         IN RAW            DEFAULT NULL)
RETURN RAW;
```

Table 10-269 DBE_RAW.OVERLAY parameters

Parameter	Description
overlay_str	Byte used for overwriting. The value cannot be NULL.
target	Source byte string to be overlaid. The value is of the RAW type and contains a maximum of 32767 bytes. The value cannot be NULL.
pos	Indicates the byte from which the overlay starts. The position of the first byte is 1. The value of **pos** must be greater than or equal to 1 and the value of **len+pos** must be less than or equal to 32767. The default value is **1**.
len	Length to be overlaid. The value of **len** must be greater than or equal to 0 and the value of **len+pos** must be less than or equal to 32767. The default value is the length of **overlay_str**.
pad	Padding byte. Only the first byte is valid. The default value is **NULL**. If the value is **NULL**, it is regarded as **0x00** by default.

- DBE_RAW.REVERSE

  Reverses RAW data by byte.

  The prototype of the DBE_RAW.REVERSE function is as follows:

```
DBE_RAW.REVERSE(
   r IN RAW
)
RETURN RAW;
```


Table 10-270 DBE_RAW.REVERSE parameters

Parameter	Description
r	The RAW value to be reversed. The maximum length is 32767. If the value is **NULL**, **NULL** is returned.

- DBE_RAW.TRANSLATE

  Converts or discards a specified byte of a RAW value.

  The prototype of the DBE_RAW.TRANSLATE function is as follows:

```
DBE_RAW.TRANSLATE(
   r        IN RAW,
   from_set IN RAW,
   to_set   IN RAW)
RETURN RAW;
```

Table 10-271 DBE_RAW.TRANSLATE parameters

Parameter	Description
r	Source byte string to be converted. The value is of the RAW type and contains a maximum of 32767 bytes. The value cannot be NULL.
from_set	Bytecode to be converted in the source byte string. The value is of the RAW type. The value cannot be NULL. The bytes in **from_set** in the source byte string are replaced with the bytes in the corresponding positions in **to_set**. If **from_set** contains multiple identical bytes, only the first byte corresponding to the byte is replaced. For example, if **r[x]=from_set[n]**, **r[x]** is replaced with **to_set[n]**. If **to_set[n]** corresponding to **from_set[n]** does not exist (that is, the number of bytes of **to_set** does not exceed **n**), **r[x]** will be discarded.
to_set	Byte code converted from the **from_set** byte. The value is of the RAW type. The value cannot be NULL.

- DBE_RAW.TRANSLITERATE

  Converts a RAW value to a NUMERIC value.

  The prototype of the DBE_RAW.TRANSLITERATE function is as follows:

```
DBE_RAW.TRANSLITERATE(
   r        IN RAW,
   from_set IN RAW DEFAULT NULL,
   to_set   IN RAW DEFAULT NULL,
   pad      IN RAW DEFAULT NULL)
RETURN RAW;
```


Table 10-272 DBE_RAW.TRANSLITERATE parameters

Parameter	Description
r	Source byte string to be converted. The value is of the RAW type and contains a maximum of 32767 bytes. The value cannot be NULL.
to_set	Byte code converted from the **from_set** byte. The value is of the RAW type. The default value is **NULL**. If the value is **NULL**, all bytes in **r** that exist in **from_set** are converted to **pad**.
from_set	Bytecode to be converted in the source byte string **r**. The value is of the RAW type. The default value is **NULL**. If **from_set** is **NULL**, all bytes in the source byte string **r** are converted into equal-length data filled by **pad**. Otherwise, the bytes in **from_set** in the source byte string **r** are replaced with the bytes in the corresponding position in **to_set**. For example, if **r[x]=from_set[n]**, **r[x]** is converted to **to_set[n]**. If **to_set[n]** does not exist, **r[x]** will be converted to **pad**.
pad	Padding byte. Only the first byte is valid. The default value is **NULL**. If the value is **NULL**, it is regarded as **0x00** by default.

- DBE_RAW.XRANGE

  Returns a RAW value containing the succession of one-byte encodings beginning and ending with the specified byte-code.

  The prototype of the DBE_RAW.XRANGE function is as follows:

```
DBE_RAW.XRANGE(
   start_byte IN RAW,
   end_byte   IN RAW)
RETURN RAW;
```

Table 10-273 DBE_RAW.XRANGE parameters

Parameter	Description
start_byte	Start byte. Only the first byte is valid. If the value is **NULL**, it is regarded as **0x00** by default.
end_byte	End byte. Only the first byte is valid. If the value is **NULL**, it is regarded as **0xFF** by default. If **end_byte** is less than **start_byte**, **end_byte** is concatenated to **0XFF**, and then **0x00** is concatenated to **start_byte**.

Permissions

The DBE_RAW schema is a system catalog. Only the system administrator has the permission to create the DBE_RAW schema during database initialization. All users have the permission to use the schema and interfaces under the schema.

Examples

```
DECLARE
v_raw RAW;
v_int INTEGER;
v_length INTEGER;
v_str VARCHAR2;
v_double BINARY_DOUBLE;
v_float FLOAT4;
v_numeric NUMERIC;
v_nvarchar2 NVARCHAR2;
BEGIN
-- Convert an INTEGER value to a RAW value.
SELECT DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW(170,1) INTO v_raw; -- 000000AA
SELECT DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW(170,2) INTO v_raw; -- AA000000
SELECT DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW(170,3) INTO v_raw; -- AA000000
-- Convert a RAW value to an INTEGER value.
SELECT
DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER(DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW(170,
1),1) INTO v_int; -- 170
SELECT
DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER(DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW(170,
2),2) INTO v_int; -- 170
SELECT
DBE_RAW.CAST_FROM_RAW_TO_BINARY_INTEGER(DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW(170,
3),3) INTO v_int; -- 170
-- Calculate the length of a RAW value.
SELECT DBE_RAW.GET_LENGTH(DBE_RAW.CAST_FROM_BINARY_INTEGER_TO_RAW(170,1)) INTO v_length;
-- 4
-- Convert a VARCHAR2 value to a RAW value.
SELECT DBE_RAW.CAST_FROM_VARCHAR2_TO_RAW('AA') INTO v_raw; -- 4141
-- Convert a RAW value to a VARCHAR2 value.
SELECT DBE_RAW.CAST_TO_VARCHAR2('4141') INTO v_str; -- AA
-- Perform bitwise OR calculation on RAW values.
SELECT DBE_RAW.BIT_OR('0000', '1111') INTO v_raw; -- 1111
-- Truncate a RAW value.
SELECT DBE_RAW.SUBSTR('ABCD', 1, 2) INTO v_raw; -- ABCD
-- Perform bitwise AND calculation on RAW values.
SELECT DBE_RAW.BIT_AND('AFF', 'FF0B') INTO v_raw; -- 0A0B
-- Perform bitwise complement calculation on RAW values.
SELECT DBE_RAW.BIT_COMPLEMENT('0AFF') INTO v_raw; -- F500
-- Perform bitwise XOR calculation on RAW values.
SELECT DBE_RAW.BIT_XOR('AFF', 'FF0B') INTO v_raw; -- F5F4
-- Convert a BINARY_DOUBLE value to a RAW value.
SELECT DBE_RAW.CAST_FROM_BINARY_DOUBLE_TO_RAW(1.0001,1) INTO v_raw; -- 3FF00068DB8BAC71
-- Convert a RAW value to a BINARY_DOUBLE value.
SELECT DBE_RAW.CAST_FROM_RAW_TO_BINARY_DOUBLE('3FF00068DB8BAC7',1) INTO v_double; -- 1.0001
-- Convert a RAW value to a FLOAT4 value.
SELECT DBE_RAW.CAST_FROM_RAW_TO_BINARY_FLOAT('40200000',1) INTO v_float; -- 2.5
-- Convert a FLOAT4 value to a RAW value.
SELECT DBE_RAW.CAST_FROM_BINARY_FLOAT_TO_RAW('2.5',1) INTO v_raw; -- 40200000
-- Convert a RAW value to a NUMERIC value.
SELECT DBE_RAW.CAST_FROM_RAW_TO_NUMBER('808002008813') INTO v_numeric; -- 2.5
-- Convert a NUMERIC value to a RAW value.
SELECT DBE_RAW.CAST_FROM_NUMBER_TO_RAW('2.5') INTO v_raw; -- 808002008813
-- Convert a RAW value to an NVARCHAR2 value.
SELECT DBE_RAW.CAST_FROM_RAW_TO_NVARCHAR2('12345678') INTO v_nvarchar2; -- \x124Vx
-- Compare RAW values.
SELECT DBE_RAW.COMPARE('ABCD','AB') INTO v_numeric; -- 2
-- Concatenate RAW values.
SELECT DBE_RAW.CONCAT('ABCD','AB') INTO v_raw; -- ABCDAB
-- Convert RAW values.
SELECT DBE_RAW.CONVERT('E695B0', 'GBK','UTF8') INTO v_raw; -- CAFD
-- Copy RAW values.
SELECT DBE_RAW.COPIES('ABCD',2) INTO v_raw; -- ABCDABCD
-- Specify the start position and length of a RAW value to be overlaid.
SELECT DBE_RAW.OVERLAY('abcef', '12345678123456', 2, 5, '9966') INTO v_raw; -- 120ABCEF999956
-- Reverse a RAW value by byte.
SELECT DBE_RAW.REVERSE('12345678') INTO v_raw; -- 78563412
-- Convert bytes of the RAW type (without padding code).
```



```
SELECT DBE_RAW.TRANSLATE('1122112233', '1133','55') INTO v_raw; -- 55225522
-- Convert bytes of the RAW type (with padding code).
SELECT DBE_RAW.TRANSLITERATE('1122112233', '55','1133','FFEE') INTO v_raw; -- 55225522FF
-- All bytes between two bytes of the RAW type.
SELECT DBE_RAW.XRANGE('00','03') INTO v_raw; -- 00010203
END;
/
ANONYMOUS BLOCK EXECUTE
```

10.12.2.13 DBE_SCHEDULER

Interface Description

The advanced package **DBE_SCHEDULER** supports more flexible creation of scheduled tasks through scheduling and programing. For details about all the supported APIs, see **Table 10-274**.

Table 10-274 DBE_SCHEDULER

Interface	Description
DBE_SCHEDULER.CREATE_JOB	Creates a scheduled task.
DBE_SCHEDULER.DROP_JOB	Deletes a scheduled task.
DBE_SCHEDULER.DROP_SINGLE_JOB	Deletes a single scheduled task.
DBE_SCHEDULER.SET_ATTRIBUTE	Sets object attributes.
DBE_SCHEDULER.RUN_JOB	Executes a scheduled task.
DBE_SCHEDULER.RUN_BACKEND_JOB	Runs a scheduled task in the background.
DBE_SCHEDULER.RUN_FOREGROUND_JOB	Runs a scheduled task in the foreground.
DBE_SCHEDULER.STOP_JOB	Stops a scheduled task.
DBE_SCHEDULER.STOP_SINGLE_JOB	Stops a single scheduled task.
DBE_SCHEDULER.GENERATE_JOB_NAME	Generates the name of a scheduled task.
DBE_SCHEDULER.CREATE_PROGRAM	Creates a program.
DBE_SCHEDULER.DEFINE_PROGRAM_ARGUMENT	Defines program parameters.

Interface	Description
DBE_SCHEDULER.DROP_PROGRAM	Deletes a program.
DBE_SCHEDULER.DROP_SINGLE_PROGRAM	Deletes a single program.
DBE_SCHEDULER.SET_JOB_ARGUMENT_VALUE	Sets the parameters of a scheduled task.
DBE_SCHEDULER.CREATE_SCHEDULE	Creates a schedule.
DBE_SCHEDULER.DROP_SCHEDULE	Deletes a schedule.
DBE_SCHEDULER.DROP_SINGLE_SCHEDULE	Deletes a single schedule.
DBE_SCHEDULER.CREATE_JOB_CLASS	Creates the class of a scheduled task.
DBE_SCHEDULER.DROP_JOB_CLASS	Deletes the class of a scheduled task.
DBE_SCHEDULER.DROP_SINGLE_JOB_CLASS	Deletes the class of a single scheduled task.
DBE_SCHEDULER.GRANT_USER_AUTHORIZATION	Grants special permissions to a user.
DBE_SCHEDULER.REVOKE_USER_AUTHORIZATION	Revokes special permissions from a user.
DBE_SCHEDULER.CREATE_CREDENTIAL	Creates a certificate.
DBE_SCHEDULER.DROP_CREDENTIAL	Destroys a certificate.
DBE_SCHEDULER.ENABLE	Enables an object.
DBE_SCHEDULER.ENABLE_SINGLE	Enables a single object.
DBE_SCHEDULER.DISABLE	Disables an object.
DBE_SCHEDULER.DISABLE_SINGLE	Disables a single object.
DBE_SCHEDULER.EVAL_CALENDAR_STRING	Analyzes the scheduling task period.

Interface	Description
DBE_SCHEDULER.EVALUATE_CALENDAR_STRING	Analyzes the scheduling task period.

- DBE_SCHEDULER.CREATE_JOB

  Creates a scheduled task.

  The prototypes of the **DBE_SCHEDULER.CREATE_JOB** function are as follows:

```
-- Scheduled tasks of an inline schedule and a program.
DBE_SCHEDULER.CREATE_JOB(
job_name TEXT,
job_type TEXT,
job_action TEXT,
number_of_arguments INTEGER             DEFAULT 0,
start_date TIMESTAMP WITH TIME ZONE     DEFAULT NULL,
repeat_interval TEXT                    DEFAULT NULL,
end_date TIMESTAMP WITH TIME ZONE       DEFAULT NULL,
job_class TEXT            DEFAULT 'DEFAULT_JOB_CLASS',
enabled BOOLEAN                         DEFAULT FALSE,
auto_drop BOOLEAN                       DEFAULT TRUE,
comments TEXT                           DEFAULT NULL,
credential_name TEXT                    DEFAULT NULL,
destination_name TEXT                   DEFAULT NULL
)

-- Reference the created scheduled tasks of the schedule and the program.
DBE_SCHEDULER.CREATE_JOB(
job_name TEXT,
program_name TEXT,
schedule_name TEXT,
job_class TEXT            DEFAULT 'DEFAULT_JOB_CLASS',
enabled BOOLEAN                         DEFAULT FALSE,
auto_drop BOOLEAN                       DEFAULT TRUE,
comments TEXT                           DEFAULT NULL,
job_style TEXT                         DEFAULT 'REGULAR',
credential_name TEXT                    DEFAULT NULL,
destination_name TEXT                   DEFAULT NULL
)

-- Reference the created program and the scheduled task of the inline schedule.
DBE_SCHEDULER.CREATE_JOB(
job_name text,
program_name TEXT,
start_date TIMESTAMP WITH TIME ZONE     DEFAULT NULL,
repeat_interval TEXT                    DEFAULT NULL,
end_date TIMESTAMP WITH TIME ZONE       DEFAULT NULL,
job_class TEXT            DEFAULT 'DEFAULT_JOB_CLASS',
enabled BOOLEAN                         DEFAULT FALSE,
auto_drop BOOLEAN                       DEFAULT TRUE,
comments TEXT                           DEFAULT NULL,
job_style TEXT                         DEFAULT 'REGULAR',
credential_name TEXT                    DEFAULT NULL,
destination_name TEXT                   DEFAULT NULL
)

-- Reference the created schedule and the scheduled task of the inline program.
DBE_SCHEDULER.CREATE_JOB(
job_name TEXT,
schedule_name TEXT,
job_type TEXT,
job_action TEXT,
number_of_arguments INTEGER             DEFAULT 0,
```



```
job_class TEXT             DEFAULT 'DEFAULT_JOB_CLASS',
enabled BOOLEAN                    DEFAULT FALSE,
auto_drop BOOLEAN                   DEFAULT TRUE,
comments TEXT                     DEFAULT NULL,
credential_name TEXT               DEFAULT NULL,
destination_name TEXT              DEFAULT NULL
)
```

NOTE

The scheduled task created through **DBE_SCHEDULER** does not conflict with the scheduled task in **DBE_TASK**.

The scheduled task created by **DBE_SCHEDULER** generates the corresponding **job_id**. However, the **job_id** is meaningless.

For the create API, the validity of input parameter types is not verified. Successful creation does not mean successful execution. You can query the execution status of the current task in the pg_job system catalog.

Table 10-275 DBE_SCHEDULER.CREATE_JOB interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job_name	text	IN	No	Name of a scheduled task.
job_type	text	IN	No	Inline program type of a scheduled task. The options are as follows: • **'PLSQL_BLOCK'**: fast anonymous stored procedure. • **'STORED_PROCEDURE'**: stored procedure that is saved. • **'EXTERNAL_SCRIPT'**: external script.
job_action	text	IN	No	Content executed by an inline program of a scheduled task.
number_of_arguments	integer	IN	No	Number of inline program parameters of a scheduled task.
program_name	text	IN	No	Name of the program referenced by a scheduled task.
start_date	timestamp with time zone	IN	Yes	Inline scheduling start time of a scheduled task.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
repeat_interval	text	IN	Yes	Inline scheduling period of a scheduled task. The options are as follows: ● 'interval N second/minute/hour/day/week/month/year' ● 'FREQ=secondly/miuntely/hourly/daily/weekly/monthy/yearly; INTERVAL=N' ● trunc(sysdate+N) + M/24 (indicating that schedule will be started in *M* hours after *N* days)
end_date	timestamp with time zone	IN	Yes	Inline scheduling expiration time of a scheduled task.
schedule_name	text	IN	No	Name of the schedule referenced by a scheduled task.
job_class	text	IN	No	Class name of a scheduled task.
enabled	boolean	IN	No	Status of a scheduled task.
auto_drop	boolean	IN	No	Automatic deletion of a scheduled task.
comments	text	IN	Yes	Comments.
job_style	text	IN	No	Behavior pattern of a scheduled task. Only **'REGULAR'** is supported.
credential_name	text	IN	Yes	Certificate name of a scheduled task.
destination_name	text	IN	Yes	Target name of a scheduled task.

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select
pg_sleep(1);', 3, false, 'test');
 create_program
----------------
```



```
(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule1', NULL, 'sysdate', NULL, 'test');
 create_schedule
-----------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_job(job_name=>'job1', program_name=>'program1',
schedule_name=>'schedule1');
 create_job
------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule1');
 drop_schedule
---------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------

(1 row)
```

NOTICE

To create a scheduled task of the EXTERNAL_SCRIPT type, the administrator must assign related permissions and certificates and the user who starts the database must have the read permission on the external script.

- DBE_SCHEDULER.DROP_JOB

  Deletes a scheduled task.

  The prototype of the **DBE_SCHEDULER.DROP_JOB** function is as follows:

```
DBE_SCHEDULER.drop_job(
job_name text,
force boolean                   default false,
defer boolean                   default false,
commit_semantics text      default 'STOP_ON_FIRST_ERROR'
)
```

  NOTE

  DBE_SCHEDULER.DROP_JOB can specify one or more tasks, or specify a task class to delete a scheduled task.

Table 10-276 DBE_SCHEDULER.DROP_JOB interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job_name	text	IN	No	Name or class of a scheduled task. You can specify one or more scheduled tasks. If you specify multiple scheduled tasks, separate them with commas (,).
force	boolean	IN	No	Specifies whether to delete a scheduled task. ● **true**: The current scheduled task is stopped and then deleted. ● **false**: The scheduled task fails to be deleted if it is running.
defer	boolean	IN	No	Specifies whether to delete a scheduled task. ● **true**: A scheduled task can be deleted after it is complete. ● **false**: The scheduled task cannot be executed and is deleted.
commit_semantics	text	IN	No	Commit rules: ● **'STOP_ON_FIRST_ERROR'**: The deletion operation performed before the first error is reported is committed. ● **'TRANSACTIONAL'**: Transaction-level commit. The deletion operation performed before an error is reported will be rolled back. ● **'ABSORB_ERRORS'**: The system attempts to bypass an error and commit the deletion operation that is performed successfully.

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select
pg_sleep(1);', 3, false, 'test');
 create_program
----------------

(1 row)
```



```
gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule1', NULL, 'sysdate', NULL, 'test');
 create_schedule
-----------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_job(job_name=>'job1', program_name=>'program1',
schedule_name=>'schedule1');
 create_job
------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule1');
 drop_schedule
---------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------

(1 row)
```

NOTICE

The **TRANSACTIONAL** option in **commit_semantic** takes effect only when **force** is set to **false**.

- DBE_SCHEDULER.DROP_SINGLE_JOB

  Deletes a scheduled task.

  The prototype of the **DBE_SCHEDULER.DROP_SINGLE_JOB** function is as follows:

```
DBE_SCHEDULER.drop_single_job(
job_name text,
force boolean                   default false,
defer boolean                   default false
)
```


Table 10-277 DBE_SCHEDULER.DROP_SINGLE_JOB interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job_name	text	IN	No	Specifies the name of a scheduled task or scheduled task class.
force	boolean	IN	No	Specifies whether to delete a scheduled task. ● **true**: The current scheduled task is stopped and then deleted. ● **false**: The scheduled task fails to be deleted if it is running.
defer	boolean	IN	No	Specifies whether to delete a scheduled task. ● **true**: A scheduled task can be deleted after it is complete. ● **false**: The scheduled task cannot be executed and is deleted.

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select
pg_sleep(1);', 0, false, 'test');
 create_program
----------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_job('job1', 'program1', '2021-07-20', 'interval ''3 minute''',
'2121-07-20', 'DEFAULT_JOB_CLASS', false, false,'test', 'style', NULL, NULL);
 create_job
------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_single_job('job1', false, false);
 drop_single_job
-----------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------

(1 row)
```

- DBE_SCHEDULER.SET_ATTRIBUTE

  Modifies the attributes of a scheduled task.

The prototypes of the **DBE_SCHEDULER.SET_ATTRIBUTE** function are as follows:

```
DBE_SCHEDULER.set_attribute(
name                text,
attribute           text,
value               boolean
)

DBE_SCHEDULER.set_attribute(
name                text,
attribute           text,
value               text
)

DBE_SCHEDULER.set_attribute(
name                text,
attribute           text,
value               timestamp
)

DBE_SCHEDULER.set_attribute(
name                text,
attribute           text,
value               timestamp with time zone
)

DBE_SCHEDULER.set_attribute(
name text,
attribute text,
value text,
value2 text                     default NULL
)
```

NOTE

name specifies any object in **DBE_SCHEDULER**.

Table 10-278 DBE_SCHEDULER.SET_ATTRIBUTE interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
name	text	IN	No	Object name.
attribute	text	IN	No	Attribute name.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
value	boolean/date/timestamp/timestamp with time zone/text	IN	No	Attribute value. The options are as follows: • Scheduled-task-related: job_type, job_action, number_of_arguments, start_date, repeat_interval, end_date, ob_class, enabled, auto_drop, comments, credential_name, destination_name, program_name, schedule_name, and job_style. • Program-related: program_action, program_type, number_of_arguments, and comments. • Scheduling-related: start_date, repeat_interval, end_date, and comments.
value2	text	IN	Yes	Additional attribute value. Reserved parameter bit. Currently, the target attribute with extra attribute values is not supported.

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select
pg_sleep(1);', 3, false, 'test');
 create_program
----------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.set_attribute('program1', 'number_of_arguments', 0);
 set_attribute
---------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.set_attribute('program1', 'program_type', 'STORED_PROCEDURE');
 set_attribute
---------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------
```



```
(1 row)
```

NOTICE

Do not use **DBE_SCHEDULER.SET_ATTRIBUTE** to leave the parameters empty.

The object name cannot be changed using **DBE_SCHEDULER.SET_ATTRIBUTE**.

- DBE_SCHEDULER.RUN_JOB

  Executes a scheduled task.

  The prototype of the **DBE_SCHEDULER.RUN_JOB** function is as follows:

```
DBE_SCHEDULER.run_job(
job_name text,
use_current_session boolean             default true
)
```

  NOTE

  DBE_SCHEDULER.RUN_JOB is used to run scheduled tasks immediately. It is independent of the scheduling of scheduled tasks and can even run at the same time.

  Table 10-279 DBE_SCHEDULER.RUN_JOB interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job_name	text	IN	No	Name of a scheduled task. You can specify one or more scheduled tasks. If you specify multiple scheduled tasks, separate them with commas (,).
use_current_session	boolean	IN	No	Specifies whether to run a scheduled task. • **true**: Use the current session to check whether the scheduled task can run properly. • **false**: Start the scheduled task in the background. The execution result is recorded in logs.

  Example:

```
gaussdb=# SELECT dbe_scheduler.create_job('job1','PLSQL_BLOCK','begin insert into test1 values(12); end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
```



```
 create_job
------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.run_job('job1', false);
 run_job
---------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)
```

NOTICE

use_current_session cannot be used to record execution results.

- DBE_SCHEDULER.RUN_BACKEND_JOB

  Runs a scheduled task in the background.

  The prototype of the **DBE_SCHEDULER.RUN_BACKEND_JOB** function is as follows:

```
DBE_SCHEDULER.run_backend_job(
job_name text
)
```

Table 10-280 DBE_SCHEDULER.RUN_BACKEND_JOB interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job_name	text	IN	No	Name of a scheduled task.

Example:

```
gaussdb=# SELECT dbe_scheduler.create_job('job1','PLSQL_BLOCK','begin insert into test1 values(12);
end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
 create_job
------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.run_backend_job('job1');
 run_backend_job
-----------------

(1 row)
```



```
gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)
```

- DBE_SCHEDULER.RUN_FOREGROUND_JOB

  Executes a scheduled task in the current session.

  Only external tasks can be executed.

  Return value: text

  The prototype of the **DBE_SCHEDULER.RUN_FOREGROUND_JOB** function is as follows:

```
DBE_SCHEDULER.run_foreground_job(
job_name text
)return text
```

  Table 10-281 DBE_SCHEDULER.RUN_FOREGROUND_JOB interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job_name	text	IN	No	Name of a scheduled task.

  Example:

```
gaussdb=# create user test1 identified by '*********';
NOTICE:  The encrypted password contains MD5 ciphertext, which is not secure.
CREATE ROLE
gaussdb=# select DBE_SCHEDULER.create_credential('cre_1', 'test1', '*********');
 create_credential
-------------------

(1 row)

gaussdb=# select DBE_SCHEDULER.create_job(job_name=>'job1', job_type=>'EXTERNAL_SCRIPT',
job_action=>'/usr/bin/pwd', enabled=>true, auto_drop=>false, credential_name => 'cre_1');
 create_job
------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.run_foreground_job('job1');
  run_foreground_job
---------------------------------
 Host key verification failed.\r+

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------
```



```
(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_credential('cre_1', false);
 drop_credential
-----------------

(1 row)

gaussdb=# drop user test1;
DROP ROLE
```

- DBE_SCHEDULER.STOP_JOB

  Stops a scheduled task.

  The prototype of the **DBE_SCHEDULER.STOP_JOB** function is as follows:

```
DBE_SCHEDULER.stop_job(
job_name text,
force boolean                   default false,
commit_semantics text               default 'STOP_ON_FIRST_ERROR'
)
```

Table 10-282 DBE_SCHEDULER.STOP_JOB interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job_name	text	IN	No	Name or class of a scheduled task. You can specify one or more scheduled tasks. If you specify multiple scheduled tasks, separate them with commas (,).
force	boolean	IN	No	Specifies whether to delete a scheduled task. • **true**: The scheduler sends a termination signal to end the task thread immediately. • **false**: The scheduler attempts to use the interrupt signal to terminate the scheduled task thread.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
commit_semantics	text	IN	No	Commit rules: • **'STOP_ON_FIRST_ERROR'**: The interrupt operation performed before the first error is reported is committed. • **'ABSORB_ERRORS'**: The system attempts to bypass an error and commit the interrupt operation that is performed successfully.

Example:

```
gaussdb=# SELECT dbe_scheduler.create_job('job1','PLSQL_BLOCK','begin insert into test1 values(12);
end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
 create_job
------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.stop_job('job1', true, 'STOP_ON_FIRST_ERROR');
 stop_job
----------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)
```

- DBE_SCHEDULER.STOP_SINGLE_JOB

  Stops a single scheduled task.

  The prototype of the **DBE_SCHEDULER.STOP_SINGLE_JOB** function is as follows:

  ```
  DBE_SCHEDULER.stop_single_job(
  job_name text,
  force boolean                    default false
  )
  ```


Table 10-283 DBE_SCHEDULER.STOP_SINGLE_JOB interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job_name	text	IN	No	Specifies the name of a scheduled task or scheduled task class.
force	boolean	IN	No	Specifies whether to delete a scheduled task. ● **true**: The scheduler sends a termination signal to end the task thread immediately. ● **false**: The scheduler attempts to use the interrupt signal to terminate the scheduled task thread.

Example:

```
gaussdb=# SELECT dbe_scheduler.create_job('job1','PLSQL_BLOCK','begin insert into test1 values(12);
end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
 create_job
------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.stop_single_job('job1', true);
 stop_single_job
-----------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)
```

- DBE_SCHEDULER.GENERATE_JOB_NAME

  Generates the name of a scheduled task.

  The prototype of the **DBE_SCHEDULER.GENERATE_JOB_NAME** function is as follows:

  ```
  DBE_SCHEDULER.generate_job_name(
  prefix text                    default 'JOB$_'
  )return text
  ```


Table 10-284 DBE_SCHEDULER.GENERATE_JOB_NAME interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
prefix	text	IN	No	Prefix of the generated name. The default value is **'JOB$_'**. Scheduled tasks that are repeatedly executed are named as follows: job$_1, job$_2, job$_3 ...

Example:

```
gaussdb=# CALL DBE_SCHEDULER.generate_job_name();
 generate_job_name
-------------------
 JOB$_1
(1 row)

gaussdb=# CALL DBE_SCHEDULER.generate_job_name();
 generate_job_name
-------------------
 JOB$_2
(1 row)

gaussdb=# CALL DBE_SCHEDULER.generate_job_name('job');
 generate_job_name
-------------------
 job3
(1 row)

gaussdb=# CALL DBE_SCHEDULER.generate_job_name('job');
 generate_job_name
-------------------
 job4
(1 row)
```

NOTICE

When DBE_SCHEDULER.GENERATE_JOB_NAME is executed for the first time, a temporary sequence is created in **public** to store the sequence number of the current name. A common user does not have the create permission in **public**. Therefore, if a common user calls the function for the first time in the current database, the function fails to be called. In this case, you need to grant the create permission in **public** to the common user or call the API as a user with the create permission to create a temporary sequence.

- DBE_SCHEDULER.CREATE_PROGRAM

  Creates a program.

The prototype of the **DBE_SCHEDULER.CREATE_PROGRAM** function is as follows:

```
DBE_SCHEDULER.create_program(
program_name text,
program_type text,
program_action text,
number_of_arguments integer          default 0,
enabled boolean                      default false,
comments text                        default NULL
)
```

Table 10-285 DBE_SCHEDULER.CREATE_PROGRAM interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
program_name	text	IN	No	Name of a program
program_type	text	IN	No	Program type. The options are as follows: ● **'PLSQL_BLOCK'**: fast anonymous stored procedure. ● **'STORED_PROCEDURE'**: stored procedure that is saved. ● **'EXTERNAL_SCRIPT'**: external script.
program_action	text	IN	No	Program operation.
number_of_arguments	integer	IN	No	Number of parameters used by the program.
enabled	boolean	IN	No	Specifies whether the program is enabled.
comments	text	IN	Yes	Comments.

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select
pg_sleep(1);', 3, false, 'test');
 create_program
----------------

(1 row)
```



```
gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------

(1 row)
```

- DBE_SCHEDULER.DEFINE_PROGRAM_ARGUMENT

  Defines program parameters.

  An API with the default value **default_value** will not convert characters to lowercase letters by default. In this version, characters are case sensitive.

  The prototype of the **DBE_SCHEDULER.DEFINE_PROGRAM_ARGUMENT** function is as follows:

```
DBE_SCHEDULER.define_program_argument(
program_name text,
argument_position integer,
argument_name text                  default NULL,
argument_type text,
out_argument boolean                 default false
)

-- With a default value --
DBE_SCHEDULER.define_program_argument(
program_name text,
argument_position integer,
argument_name text                  default NULL,
argument_type text,
default_value text,
out_argument boolean                 default false
)
```

Table 10-286 DBE_SCHEDULER.DEFINE_PROGRAM_ARGUMENT interface parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
program_name	text	IN	No	Name of a program
argument_position	integer	IN	No	Parameter location
argument_name	text	IN	No	Parameter name
argument_type	text	IN	No	Parameter type
default_value	text	IN	No	Default value

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
out_argument	boolean	IN	No	Reserved parameter

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select
pg_sleep(1);', 2, false, 'test');
create_program
----------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.define_program_argument('program1', 1, 'pa1', 'type1', false);
 define_program_argument
-------------------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.define_program_argument('program1', 1, 'pa1', 'type1', 'value1',
false);
 define_program_argument
-------------------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------

(1 row)
```

- DBE_SCHEDULER.DROP_PROGRAM

  Deletes a program.

  The prototype of the **DBE_SCHEDULER.DROP_PROGRAM** function is as follows:

  ```
  DBE_SCHEDULER.drop_program(
  program_name text,
  force boolean                    default false
  )
  ```


Table 10-287 DBE_SCHEDULER.DROP_PROGRAM interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
program_name	text	IN	No	Name of a program
force	boolean	IN	No	Specifies whether to delete a program. • **true**: Before a program is deleted, all jobs that use the program are disabled. • **false**: The program cannot be referenced by any job. Otherwise, an error is sent.

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select
pg_sleep(1);', 2, false, 'test');
create_program
----------------


(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------


(1 row)
```

- DBE_SCHEDULER.DROP_SINGLE_PROGRAM

  Deletes a single program.

  The prototype of the **DBE_SCHEDULER.DROP_SINGLE_PROGRAM** function is as follows:

  ```
  DBE_SCHEDULER.drop_single_program(
  program_name text,
  force boolean                    default false
  )
  ```


Table 10-288 DBE_SCHEDULER.DROP_SINGLE_PROGRAM interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
program_name	text	IN	No	Name of a program
force	boolean	IN	No	Specifies whether to delete a program. ● **true**: Before a program is deleted, all jobs that use the program are disabled. ● **false**: The program cannot be referenced by any job. Otherwise, an error is sent.

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'STORED_PROCEDURE', 'select
pg_sleep(1);', 2, false, 'test');
create_program
----------------


(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_single_program('program1', false);
 drop_single_program
---------------------


(1 row)
```

- DBE_SCHEDULER.SET_JOB_ARGUMENT_VALUE

  Sets the parameters of a scheduled task. The **argument_value** can be left empty.

  The prototype of the **DBE_SCHEDULER.SET_JOB_ARGUMENT_VALUE** function is as follows:

```
DBE_SCHEDULER.set_job_argument_value(
job_name text,
argument_position integer,
argument_value text
)

DBE_SCHEDULER.set_job_argument_value(
job_name text,
argument_name text,
argument_value text
)
```


Table 10-289 DBE_SCHEDULER.SET_JOB_ARGUMENT_VALUE interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job_name	text	IN	No	Name of a scheduled task.
argument_position	integer	IN	No	Parameter location.
argument_name	text	IN	Yes	Parameter name.
argument_value	text	IN	Yes	Parameter value.

Example:

```
gaussdb=# CALL dbe_scheduler.create_job('job1','EXTERNAL_SCRIPT','begin insert into test1
values(12); end;',2,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
 create_job
------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.set_job_argument_value('job1', 1, 'value1');
 set_job_argument_value
------------------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)
```

- DBE_SCHEDULER.CREATE_SCHEDULE

  Creates a schedule.

  The prototype of the **DBE_SCHEDULER.CREATE_SCHEDULE** function is as follows:

```
DBE_SCHEDULER.create_schedule(
schedule_name text,
start_date timestamp with time zone      default NULL,
repeat_interval text,
end_date timestamp with time zone        default NULL,
comments text                            default NULL
)
```


Table 10-290 DBE_SCHEDULER.CREATE_SCHEDULE interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
schedule_name	text	IN	No	Name of a schedule.
start_date	timestamp with time zone	IN	Yes	Start time of a schedule.
repeat_interval	text	IN	No	Repetition frequency of a schedule.
end_date	timestamp with time zone	IN	Yes	End time of a schedule.
comments	text	IN	Yes	Comments.

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule1', sysdate, 'sysdate + 3 / (24 * 60 * 60)',
null, 'test1');
 create_schedule
-----------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule2', sysdate, 'FREQ=DAILY; BYHOUR=6;',
null, 'test1');
 create_schedule
-----------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule3', sysdate, 'FREQ=DAILY; BYHOUR=6;');
 create_schedule
-----------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule1');
 drop_single_schedule
----------------------

(1 row)
```



```
gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule2', false);
 drop_single_schedule
----------------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule3', true);
 drop_single_schedule
----------------------

(1 row)
```

- DBE_SCHEDULER.DROP_SCHEDULE

  Deletes a schedule.

  The prototype of the **DBE_SCHEDULER.DROP_SCHEDULE** function is as follows:

```
DBE_SCHEDULER.drop_schedule(
schedule_name text,
force boolean                    default false
)
```

Table 10-291 DBE_SCHEDULER.DROP_SCHEDULE interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
schedule_name	text	IN	No	Name of a schedule.
force	boolean	IN	No	Specifies whether to delete a schedule. • **true**: Any jobs or windows that use this schedule are disabled before the schedule is deleted. • **false**: The schedule cannot be referenced by any job or window. Otherwise, an error occurs.

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule1', sysdate, 'sysdate + 3 / (24 * 60 * 60)',
null, 'test1');
 create_schedule
-----------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule2', sysdate, 'FREQ=DAILY; BYHOUR=6;',
null, 'test1');
 create_schedule
-----------------
```



```
(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule3', sysdate, 'FREQ=DAILY; BYHOUR=6;');
 create_schedule
-----------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule1');
 drop_single_schedule
----------------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule2', false);
 drop_single_schedule
----------------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_schedule('schedule3', true);
 drop_single_schedule
----------------------

(1 row)
```

- DBE_SCHEDULER.DROP_SINGLE_SCHEDULE

  Deletes a single schedule.

  The prototype of the **DBE_SCHEDULER.DROP_SINGLE_SCHEDULE** function is as follows:

```
DBE_SCHEDULER.drop_single_schedule(
schedule_name text,
force boolean                   default false
)
```

Table 10-292 DBE_SCHEDULER.DROP_SINGLE_SCHEDULE interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
schedule_name	text	IN	No	Name of a schedule.
force	boolean	IN	No	Specifies whether to delete a schedule. • **true**: Any jobs or windows that use this schedule are disabled before the schedule is deleted. • **false**: The schedule cannot be referenced by any job or window. Otherwise, an error occurs.

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule1', sysdate, 'sysdate + 3 / (24 * 60 * 60)',
null, 'test1');
 create_schedule
-----------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule2', sysdate, 'FREQ=DAILY; BYHOUR=6;',
null, 'test1');
 create_schedule
-----------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_schedule('schedule3', sysdate, 'FREQ=DAILY; BYHOUR=6;');
 create_schedule
-----------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_single_schedule('schedule1');
 drop_single_schedule
----------------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_single_schedule('schedule2', false);
 drop_single_schedule
----------------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_single_schedule('schedule3', true);
 drop_single_schedule
----------------------

(1 row)
```

- DBE_SCHEDULER.CREATE_JOB_CLASS

  Creates the class of a scheduled task.

  The prototype of the **DBE_SCHEDULER.CREATE_JOB_CLASS** function is as follows:

```
DBE_SCHEDULER.create_job_class(
job_class_name text,
resource_consumer_group text           default NULL,
service text                        default NULL,
logging_level integer                  default 0,
log_history integer                   default NULL,
comments text                          default NULL
)
```


Table 10-293 DBE_SCHEDULER.CREATE_JOB_CLASS interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job_class_name	text	IN	No	Name of a scheduled task class.
resource_consumer_group	text	IN	Yes	Inline resource consumer group of a scheduled task class.
service	text	IN	Yes	Inline database service of a scheduled task class.
logging_level	integer	IN	No	Number of scheduled task records.
log_history	integer	IN	Yes	Number of days for storing scheduled task records.
comments	text	IN	Yes	Comments.

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_job_class(job_class_name => 'jc1', resource_consumer_group
=> '123');
 create_job_class
------------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job_class('jc1', false);
 drop_job_class
----------------

(1 row)
```

- DBE_SCHEDULER.DROP_JOB_CLASS

  Deletes the class of a scheduled task.

  The prototype of the **DBE_SCHEDULER.DROP_JOB_CLASS** function is as follows:

```
DBE_SCHEDULER.drop_job_class(
job_class_name text,
force boolean                    default false
)
```


Table 10-294 DBE_SCHEDULER.DROP_JOB_CLASS interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job_class_name	text	IN	No	Name of a scheduled task class.
force	boolean	IN	No	Specifies whether to delete a scheduled task class. • **true**: Jobs of this class will be disabled, and another class will be set as the default class. Only when such setting is successful, this class will be deleted. • **false**: The class to be deleted cannot be referenced by any job. Otherwise, an error occurs.

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_job_class(job_class_name => 'jc1', resource_consumer_group
=> '123');
 create_job_class
------------------


(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job_class('jc1', false);
 drop_job_class
----------------


(1 row)
```

- DBE_SCHEDULER.DROP_SINGLE_JOB_CLASS

  Deletes the class of a single scheduled task.

  The prototype of the **DBE_SCHEDULER.DROP_SINGLE_JOB_CLASS** function is as follows:

  ```
  DBE_SCHEDULER.drop_single_job_class(
  job_class_name text,
  force boolean                  default false
  )
  ```


Table 10-295 DBE_SCHEDULER.DROP_SINGLE_JOB_CLASS interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job_class_name	text	IN	No	Name of a scheduled task class.
force	boolean	IN	No	Specifies whether to delete a scheduled task class. ● **true**: Jobs of this class will be disabled, and another class will be set as the default class. Only when such setting is successful, this class will be deleted. ● **false**: The class to be deleted cannot be referenced by any job. Otherwise, an error occurs.

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_job_class(job_class_name => 'jc1', resource_consumer_group
=> '123');
 create_job_class
------------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_single_job_class('jc1', false);
 drop_job_class
----------------

(1 row)
```

- DBE_SCHEDULER.GRANT_USER_AUTHORIZATION

  Grants the scheduled task permissions to the database user. The user who calls this function must have the SYSADMIN permission.

  The prototype of the **DBE_SCHEDULER.GRANT_USER_AUTHORIZATION** function is as follows:

```
DBE_SCHEDULER.grant_user_authorization(
username              text,
privilege             text
)
```


Table 10-296 DBE_SCHEDULER.GRANT_USER_AUTHORIZATION interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
username	text	IN	No	Database username.
privilege	text	IN	No	Scheduled task permission.

Example:

```
gaussdb=# create user user1 password '1*s*****';
NOTICE:  The encrypted password contains MD5 ciphertext, which is not secure.
CREATE ROLE
gaussdb=# CALL DBE_SCHEDULER.grant_user_authorization('user1', 'create job');
 grant_user_authorization
--------------------------


(1 row)
gaussdb=# drop user user1;
DROP ROLE
```

- DBE_SCHEDULER.REVOKE_USER_AUTHORIZATION

  Revokes the scheduled task permissions from the database user. The user who calls this function must have the SYSADMIN permission.

  The prototype of the **DBE_SCHEDULER.REVOKE_USER_AUTHORIZATION** function is as follows:

```
DBE_SCHEDULER.revoke_user_authorization(
username              text,
privilege             text
)
```

Table 10-297 DBE_SCHEDULER.REVOKE_USER_AUTHORIZATION interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
username	text	IN	No	Database username.
privilege	text	IN	No	Scheduled task permission.

Example:

```
gaussdb=# create user user1 password '1*s*****';
NOTICE:  The encrypted password contains MD5 ciphertext, which is not secure.
CREATE ROLE
gaussdb=# CALL DBE_SCHEDULER.grant_user_authorization('user1', 'create job');
 grant_user_authorization
--------------------------


(1 row)

gaussdb=# CALL DBE_SCHEDULER.revoke_user_authorization('user1', 'create job');
 revoke_user_authorization
---------------------------


(1 row)
gaussdb=# drop user user1;
DROP ROLE
```

- DBE_SCHEDULER.CREATE_CREDENTIAL

  Creates an authorization certificate. The user who calls this function must have the SYSADMIN permission.

  The prototype of the **DBE_SCHEDULER.CREATE_CREDENTIAL** function is as follows:

```
DBE_SCHEDULER.create_credential(
credential_name         text,
username                text,
password                text            default NULL,
database_role           text            default NULL,
windows_domain          text            default NULL,
comments                text            default NULL
)
```

Table 10-298 DBE_SCHEDULER.CREATE_CREDENTIAL interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
credential_name	text	IN	No	Name of the authorization certificate.
username	text	IN	No	Database username.
password	text	IN	Yes	User password.
database_role	text	IN	Yes	Database system permission.
windows_domain	text	IN	Yes	Domain to which a Windows user belongs.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
comments	text	IN	Yes	Comments.

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_credential('cre_1', 'user1', '');
 create_credential
-------------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_credential('cre_1', false);
 drop_credential
-----------------

(1 row)
```

NOTICE

The **password** parameter of **DBE_SCHEDULER.CREATE_CREDENTIAL** must be set to **NULL** or **'******'**. This parameter is used only for compatibility and does not indicate any actual meaning. Do not use the OS username corresponding to the installation user to create a certificate.

- DBE_SCHEDULER.DROP_CREDENTIAL

  Destroys an authorization certificate. The user who calls this function must have the SYSADMIN permission.

  The prototype of the **DBE_SCHEDULER.DROP_CREDENTIAL** function is as follows:

```
DBE_SCHEDULER.drop_credential(
credential_name  text,
force            boolean default false
)
```


Table 10-299 DBE_SCHEDULER.DROP_CREDENTIAL interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
credential_name	text	IN	No	Name of the authorization certificate.
force	boolean	IN	No	Specifies whether to delete the authorization certificate. ● **true**: The certificate will be deleted no matter whether it is referenced by any job. ● **false**: No job can reference the certificate. Otherwise, an error occurs.

Example:

```
gaussdb=# CALL DBE_SCHEDULER.create_credential('cre_1', 'user1', '');
 create_credential
-------------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_credential('cre_1', false);
 drop_credential
-----------------

(1 row)
```

- DBE_SCHEDULER.ENABLE

  Enables an object.

  The prototype of the **DBE_SCHEDULER.ENABLE** function is as follows:

```
DBE_SCHEDULER.enable(
name text,
commit_semantics text                default 'STOP_ON_FIRST_ERROR'
)
```


Table 10-300 DBE_SCHEDULER.ENABLE interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
name	text	IN	No	Object name. You can specify one or more objects. If you specify multiple objects, separate them with commas (,).
commit_semantics	text	IN	No	Commit rules. The following types are supported: ● **'STOP_ON_FIRST_ERROR'**: The enabling operation performed before the first error is reported is committed. ● **'TRANSACTIONAL'**: Transaction-level commit. The enabling operation performed before an error is reported will be rolled back. ● **'ABSORB_ERRORS'**: The system attempts to bypass an error and commit the enabling operation that is performed successfully.

Example:

```
gaussdb=# CALL dbe_scheduler.create_job('job1','PLSQL_BLOCK','begin insert into test1 values(12);
end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
lse, 'test');
 create_job
------------


(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'stored_procedure', 'insert into
tb_job_test(key) values(null);', 0, false, '');
 create_program
----------------


(1 row)

gaussdb=# CALL DBE_SCHEDULER.enable('job1');
 enable
--------


(1 row)

gaussdb=# CALL DBE_SCHEDULER.enable('program1', 'STOP_ON_FIRST_ERROR');
 enable
--------
```



```
(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------

(1 row)
```

- DBE_SCHEDULER.ENABLE_SINGLE

  Enables a single object.

  The prototype of the **DBE_SCHEDULER.ENABLE_SINGLE** function is as follows:

  ```
  DBE_SCHEDULER.enable_single(
  name text
  )
  ```

  Table 10-301 DBE_SCHEDULER.ENABLE_SINGLE interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
name	text	IN	No	Object name.

  Example:

  ```
  gaussdb=# CALL dbe_scheduler.create_job('job1','PLSQL_BLOCK','begin insert into test1 values(12);
  end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
  lse, 'test');
   create_job
  ------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.enable_single('job1');
   enable_single
  ---------------

  (1 row)

  gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
   drop_job
  ----------

  (1 row)
  ```

- DBE_SCHEDULER.DISABLE

Disables multiple objects. The value of name is a character string separated by commas (,). Each character string separated by commas (,) is an object. Operations are synchronized only when operation synchronization is enabled.

The prototype of the **DBE_SCHEDULER.DISABLE** function is as follows:

```
DBE_SCHEDULER.disable(
name text,
force boolean                   default false,
commit_semantics text               default 'STOP_ON_FIRST_ERROR'
)
```

Table 10-302 DBE_SCHEDULER.DISABLE interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
name	text	IN	No	Object name.
force	boolean	IN	No	Specifies whether to disable an object. ● **true**: The object is disabled regardless of whether other objects depend on it. ● **false**: No object can depend on the object. Otherwise, an error occurs.
commit_semantics	text	IN	No	Commit rules. The following types are supported: ● **'STOP_ON_FIRST_ERROR'**: The disabling operation performed before the first error is reported is committed. ● **'TRANSACTIONAL'**: Transaction-level commit. The disabling operation performed before an error is reported will be rolled back. ● **'ABSORB_ERRORS'**: The system attempts to bypass an error and commit the disabling operation that is performed successfully.

Example:

```
gaussdb=# CALL dbe_scheduler.create_job('job1','PLSQL_BLOCK','begin insert into test1 values(12);
end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
 create_job
------------
```



```
(1 row)

gaussdb=# CALL DBE_SCHEDULER.create_program('program1', 'stored_procedure', 'insert into
tb_job_test(key) values(null);', 0, false, '');
 create_program
----------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.disable('job1');
 disable
---------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.disable('program1', false, 'STOP_ON_FIRST_ERROR');
 disable
---------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_program('program1', false);
 drop_program
--------------

(1 row)
```

- DBE_SCHEDULER.DISABLE_SINGLE

  Disables a single object.

  The prototype of the **DBE_SCHEDULER.DISABLE_SINGLE** function is as follows:

```
DBE_SCHEDULER.disable_single(
name text,
force boolean                    default false
)
```

Table 10-303 DBE_SCHEDULER.DISABLE_SINGLE interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
name	text	IN	No	Object name.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
force	boolean	IN	No	Specifies whether to disable an object. ● **true**: The object is disabled regardless of whether other objects depend on it. ● **false**: No object can depend on the object. Otherwise, an error occurs.

Example:

```
gaussdb=# CALL dbe_scheduler.create_job('job1','PLSQL_BLOCK','begin insert into test1 values(12);
end;',0,null,null,null,'DEFAULT_JOB_CLASS',false,false,null,null,null);
 create_job
------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.disable_single('job1', false);
 disable_single
----------------

(1 row)

gaussdb=# CALL DBE_SCHEDULER.drop_job('job1', true, false, 'STOP_ON_FIRST_ERROR');
 drop_job
----------

(1 row)
```

- DBE_SCHEDULER.EVAL_CALENDAR_STRING

  Analyzes the scheduling task period.

  Return type: timestamp with time zone

  The prototype of the **DBE_SCHEDULER.EVAL_CALENDAR_STRING** function is as follows:

  ```
  DBE_SCHEDULER.evaluate_calendar_string(
  IN calendar_string text,
  IN start_date timestamp with time zone,
  IN return_date_after timestamp with time zone
  )return timestamp with time zone
  ```


Table 10-304 DBE_SCHEDULER.EVAL_CALENDAR_STRING interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
calendar_string	text	IN	No	Date string of a scheduled task.
start_date	timestamp with time zone	IN	No	Start time of a scheduled task.
return_date_after	timestamp with time zone	IN	No	Date when a scheduled task is returned.

Example:

```
gaussdb=# CALL DBE_SCHEDULER.eval_calendar_string('FREQ=DAILY; BYHOUR=6;', sysdate, sysdate);
  eval_calendar_string
------------------------
 2023-09-16 06:05:55+08
(1 row)
```

- DBE_SCHEDULER.EVALUATE_CALENDAR_STRING

  Analyzes the scheduling task period.

  Return type: timestamp with time zone

  The prototype of the **DBE_SCHEDULER.EVALUATE_CALENDAR_STRING** function is as follows:

  ```
  DBE_SCHEDULER.evaluate_calendar_string(
  IN calendar_string text,
  IN start_date timestamp with time zone,
  IN return_date_after timestamp with time zone,
  OUT next_run_date timestamp with time zone
  )return timestamp with time zone
  ```


Table 10-305 DBE_SCHEDULER.EVALUATE_CALENDAR_STRING interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
calendar_string	text	IN	No	Date string of a scheduled task.
start_date	timestamp with time zone	IN	No	Start time of a scheduled task.
return_date_after	timestamp with time zone	IN	No	Date when a scheduled task is returned.
next_run_date	timestamp with time zone	OUT	No	The next date when a scheduled task is returned.

Example:

```
gaussdb=# CREATE OR REPLACE PROCEDURE pr1(calendar_str text) as
 DECLARE
    start_date        timestamp with time zone;
    return_date_after timestamp with time zone;
    next_run_date     timestamp with time zone;
 BEGIN
    start_date := '2003-2-1 10:30:00.111111+8'::timestamp with time zone;
    return_date_after := start_date;
    DBE_SCHEDULER.evaluate_calendar_string(
        calendar_str,
        start_date, return_date_after, next_run_date);
    DBE_OUTPUT.PRINT_LINE('next_run_date: ' || next_run_date);
    return_date_after := next_run_date;
 END;
 /
CREATE PROCEDURE
gaussdb=# CALL pr1('FREQ=hourly;INTERVAL=2;BYHOUR=6,10;BYMINUTE=0;BYSECOND=0');
next_run_date: 2003-02-02 06:00:00+08
 pr1
-----

(1 row)
```


10.12.2.14 DBE_SESSION

Interface Description

Table 10-306 provides all interfaces supported by the **DBE_SESSION** package. DBE_SESSION takes effect at the session level.

Table 10-306 DBE_SESSION

Interface	Description
DBE_SESSION.SET_CONTEXT	Sets the value of an attribute in a specified context.
DBE_SESSION.CLEAR_CONTEXT	Clears the value of an attribute in a specified context.
DBE_SESSION.SEARCH_CONTEXT	Queries the value of an attribute in a specified context.
DBE_SESSION.MODIFY_PACKAGE_STATE	Changes the PL/SQL status of the current session.

- DBE_SESSION.SET_CONTEXT

  Sets the value of an attribute in a specified namespace (context). The prototype of the **DBE_SESSION.SET_CONTEXT** function is as follows:

```
DBE_SESSION.SET_CONTEXT(
    namespace text,
    attribute text,
    value text
)returns void;
```

Table 10-307 DBE_SESSION.SET_CONTEXT interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
namespace	TEXT	IN	No	Name of the context to be set. If the context does not exist, create a context. The value contains a maximum of 128 bytes. If the value exceeds 128 bytes, it will be truncated.
attribute	TEXT	IN	No	Attribute name. The value contains a maximum of 1024 bytes. If the value exceeds 1024 bytes, it will be truncated.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
value	TEXT	IN	No	Name of the value to be set. The value contains a maximum of 1024 bytes. If the value exceeds 1024 bytes, it will be truncated.

- DBE_SESSION.CLEAR_CONTEXT

  Clears the value of an attribute in a specified namespace (context). The prototype of the **DBE_SESSION.CLEAR_CONTEXT** function is as follows:

```
DBE_SESSION.CLEAR_CONTEXT (
   namespace text,
   client_identifier text default null,
   attribute text default null
)returns void ;
```

Table 10-308 DBE_SESSION.CLEAR_CONTEXT interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
namespace	TEXT	IN	No	Context specified by the user. The value contains a maximum of 128 bytes. If the value exceeds 128 bytes, it will be truncated.
client_identifier	TEXT	IN	Yes	Client authentication. The default value is **null**. Generally, you do not need to manually set this parameter.
attribute	TEXT	IN	Yes	Attribute to be cleared. The default value is **null**, indicating that all attributes of the specified context are cleared. The value contains a maximum of 1024 bytes. If the value exceeds 1024 bytes, it will be truncated. Note: To ensure forward compatibility, if the parameter value is **'null'**, all attributes of the specified context are cleared.

- DBE_SESSION.SEARCH_CONTEXT

  Queries the value of an attribute in a specified namespace (context). The prototype of the **DBE_SESSION.SEARCH_CONTEXT** function is as follows:

```
DBE_SESSION.SEARCH_CONTEXT (
   namespace text,
```



```
    attribute text
)returns text;
```

Table 10-309 DBE_SESSION.SEARCH_CONTEXT interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
namespace	TEXT	IN	No	Context specified by the user. The value contains a maximum of 128 bytes. If the value exceeds 128 bytes, it will be truncated.
attribute	TEXT	IN	No	Attribute to be searched for. The value contains a maximum of 1024 bytes. If the value exceeds 1024 bytes, it will be truncated.

Example:

```
DECLARE
    a text;
BEGIN
    DBE_SESSION.set_context('test', 'gaussdb', 'one');   --Set the gaussdb attribute in the test context to one.
    a := DBE_SESSION.search_context('test', 'gaussdb');
    DBE_OUTPUT.PRINT_LINE(a);
    DBE_SESSION.clear_context('test', 'test','gaussdb');
END;
/
-- Expected result:
one
ANONYMOUS BLOCK EXECUTE
```

- DBE_SESSION.MODIFY_PACKAGE_STATE

  Changes the PL/SQL status of the current session. The prototype of the DBE_SESSION.MODIFY_PACKAGE_STATE function is:

```
DBE_SESSION.MODIFY_PACKAGE_STATE (
    flags int);
```


Table 10-310 DBE_SESSION.MODIFY_PACKAGE_STATE interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
flags	INT	IN	No	Bit flag of the operation performed on the PL/SQL. When **flags** is set to **1**, the session cache of the PL/SQL that is currently running in the session is released after the top-level PL/SQL is executed. The current value of any package is cleared globally and the cached cursor is closed. On subsequent use, PL/SQL re-instantiates and re-initializes the package globally. Other bit flags are not supported.

Example:

```
CALL dbe_session.modify_package_state(1);
 modify_package_state
----------------------

(1 row)
```

10.12.2.15 DBE_SQL

Data Types

- DBE_SQL.DESC_REC

  This type is a composite type and is used to store the description of the SQL_DESCRIBE_COLUMNS API.

  The prototype of the DBE_SQL.DESC_REC type is as follows:

```
CREATE TYPE DBE_SQL.DESC_REC AS (
    col_type            int,
    col_max_len         int,
    col_name            VARCHAR2(32),
    col_name_len        int,
    col_schema_name     VARCHAR2(32),
    col_schema_name_len int,
    col_precision       int,
    col_scale           int,
    col_charsetid       int,
    col_charsetform     int,
    col_null_ok         BOOLEAN
);
```

- DBE_SQL.DESC_TAB

  This type is the TABLE type of DESC_REC and is implemented through the TABLE OF syntax.

  The prototype of the DBE_SQL.DESC_TAB type is as follows:

```
CREATE TYPE DBE_SQL.DESC_TAB AS TABLE OF DBE_SQL.DESC_REC INDEX BY INTEGER;
```


- DBE_SQL.DATE_TABLE

  This type is the TABLE type of DATE and is implemented through the TABLE OF syntax.

  The prototype of the DBE_SQL.DATE_TABLE type is as follows:

  ```
  CREATE TYPE DBE_SQL.DATE_TABLE AS TABLE OF DATE INDEX BY INTEGER;
  ```

- DBE_SQL.NUMBER_TABLE

  This type is the TABLE type of NUMBER and is implemented through the TABLE OF syntax.

  The prototype of the DBE_SQL.NUMBER_TABLE type is as follows:

  ```
  CREATE TYPE DBE_SQL.NUMBER_TABLE AS TABLE OF NUMBER INDEX BY INTEGER;
  ```

- DBE_SQL.VARCHAR2_TABLE

  This type is the TABLE type of VARCHAR2 and is implemented through the TABLE OF syntax.

  The prototype of the DBE_SQL.VARCHAR2_TABLE type is as follows:

  ```
  CREATE TYPE DBE_SQL.VARCHAR2_TABLE AS TABLE OF VARCHAR2(32767) INDEX BY INTEGER;
  ```

- DBE_SQL.BLOB_TABLE

  This type is the TABLE type of BLOB and is implemented through the TABLE OF syntax.

  The prototype of the DBE_SQL.BLOB_TABLE type is as follows:

  ```
  CREATE TYPE DBE_SQL.BLOB_TABLE AS TABLE OF BLOB INDEX BY INTEGER;
  ```

API Description

Table 10-311 lists APIs supported by the **DBE_SQL** package.

Table 10-311 DBE_SQL

API	Description
DBE_SQL.REGISTER_CONTEXT	Opens a cursor.
DBE_SQL.SQL_UNREGISTER_CONTEXT	Closes an open cursor.
DBE_SQL.SQL_SET_SQL	Passes a set of SQL statements or anonymous blocks to a cursor.
DBE_SQL.SQL_RUN	Executes SQL statements or anonymous blocks in a given cursor.
DBE_SQL.NEXT_ROW	Reads a row of cursor data.
DBE_SQL.SET_RESULT_TYPE	Dynamically defines a column.
DBE_SQL.SET_RESULT_TYPE_CHAR	Dynamically defines a column of the CHAR type.
DBE_SQL.SET_RESULT_TYPE_INT	Dynamically defines a column of the INT type.

API	Description
DBE_SQL.SET_RESULT_TYPE_LONG	Dynamically defines a column of the LONG type.
DBE_SQL.SET_RESULT_TYPE_RAW	Dynamically defines a column of the RAW type.
DBE_SQL.SET_RESULT_TYPE_BYTEA	Dynamically defines a column of the BYTEA type.
DBE_SQL.SET_RESULT_TYPE_TEXT	Dynamically defines a column of the TEXT type.
DBE_SQL.SET_RESULT_TYPE_UNKNOWN	Dynamically defines a column of an unknown type.
DBE_SQL.GET_RESULT	Reads a dynamically defined column value.
DBE_SQL.GET_RESULT_CHAR	Reads a dynamically defined column value of the CHAR type.
DBE_SQL.GET_RESULT_INT	Reads a dynamically defined column value of the INT type.
DBE_SQL.GET_RESULT_LONG	Reads a dynamically defined column value of the LONG type.
DBE_SQL.GET_RESULT_RAW	Reads a dynamically defined column value of the RAW type.
DBE_SQL.GET_RESULT_BYTEA	Reads a dynamically defined column value of the BYTEA type.
DBE_SQL.GET_RESULT_TEXT	Reads a dynamically defined column value of the TEXT type.
DBE_SQL.GET_RESULT_UNKNOWN	Reads a dynamically defined column value of an unknown type.
DBE_SQL.DBE_SQL_GET_RESULT_CHAR	Reads a dynamically defined column value of the CHAR type.
DBE_SQL.DBE_SQL_GET_RESULT_LONG	Reads a dynamically defined column value of the LONG type.
DBE_SQL.DBE_SQL_GET_RESULT_RAW	Reads a dynamically defined column value of the RAW type.
DBE_SQL.IS_ACTIVE	Checks whether a cursor is opened.
DBE_SQL.LAST_ROW_COUNT	Returns the cumulative count of obtained rows.

API	Description
DBE_SQL.RUN_AND_NEXT	Reads data of a cursor after a set of dynamically defined operations are performed on the cursor.
DBE_SQL.SQL_BIND_VARIABLE	Binds a value to a variable in a statement.
DBE_SQL.SQL_BIND_ARRAY	Binds a group of values to a variable in a statement.
DBE_SQL.SET_RESULT_TYPE_INTS	Dynamically defines a column of the INT array type.
DBE_SQL.SET_RESULT_TYPE_TEXTS	Dynamically defines a column of the TEXT array type.
DBE_SQL.SET_RESULT_TYPE_RAWS	Dynamically defines a column of the RAW array type.
DBE_SQL.SET_RESULT_TYPE_BYTEAS	Dynamically defines a column of the BYTEA array type.
DBE_SQL.SET_RESULT_TYPE_CHARS	Dynamically defines a column of the CHAR array type.
DBE_SQL.SET_RESULTS_TYPE	Dynamically defines a column of the array type.
DBE_SQL.GET_RESULTS_INT	Reads a dynamically defined column value of the INT array type.
DBE_SQL.GET_RESULTS_TEXT	Reads a dynamically defined column value of the TEXT array type.
DBE_SQL.GET_RESULTS_RAW	Reads a dynamically defined column value of the RAW array type.
DBE_SQL.GET_RESULTS_BYTEA	Reads a dynamically defined column value of the BYTEA array type.
DBE_SQL.GET_RESULTS_CHAR	Reads a dynamically defined column value of the CHAR array type.
DBE_SQL.GET_RESULTS	Reads a dynamically defined column value.
DBE_SQL.SQL_DESCRIBE_COLUMNS	Describes the column information read by the cursor.

API	Description
DBE_SQL.DESCRIBE_COLUMNS	Describes the column information read by the cursor.
DBE_SQL.BIND_VARIABLE	Binds parameters.
DBE_SQL.SQL_SET_RESULTS_TYPE_C	Dynamically defines a column of the array type.
DBE_SQL.SQL_GET_VALUES_C	Reads a dynamically defined column value.
DBE_SQL.GET_VARIABLE_RESULT	Reads the return value of an SQL statement.
DBE_SQL.GET_VARIABLE_RESULT_CHAR	Reads the return value (of the CHAR type) of an SQL statement.
DBE_SQL.GET_VARIABLE_RESULT_RAW	Reads the return value (of the RAW type) of an SQL statement.
DBE_SQL.GET_VARIABLE_RESULT_TEXT	Reads the return value (of the TEXT type) of an SQL statement.
DBE_SQL.GET_VARIABLE_RESULT_INT	Reads the return value (of the INT type) of an SQL statement.
DBE_SQL.GET_ARRAY_RESULT_TEXT	Reads the return value (of the TEXT array type) of an SQL statement.
DBE_SQL.GET_ARRAY_RESULT_RAW	Reads the return value (of the RAW array type) of an SQL statement.
DBE_SQL.GET_ARRAY_RESULT_CHAR	Reads the return value (of the CHAR array type) of an SQL statement.
DBE_SQL.GET_ARRAY_RESULT_INT	Reads the return value (of the INT array type) of an SQL statement.
DBE_SQL.SQL_SET_TABLEOF_RESULTS_TYPE_C	Dynamically defines a column of the tableof type.
DBE_SQL.SQL_GET_TABLEOF_VALUES_C	Reads a dynamically defined column value of the tableof type.

NOTE

- You are advised to use **dbe_sql.set_result_type** and **dbe_sql.get_result** to define columns.
- If the size of the result set is greater than the value of **work_mem**, the result set will be spilled to a disk temporarily. The value of **work_mem** must be no greater than 512 MB.

- DBE_SQL.REGISTER_CONTEXT

  This function opens a cursor, which is the prerequisite for the subsequent **dbe_sql** operations. This function does not transfer any parameter. It automatically generates cursor IDs in an ascending order and returns values to integer variables.

  CAUTION

  Cursors opened by DBE_SQL are session-level variables. Cross-session calling of opened cursors (such as autonomous transactions) is not supported. If a cross-session cursor is called, the behavior is unpredictable.

  The prototype of the DBE_SQL.REGISTER_CONTEXT function is as follows:

  ```
  DBE_SQL.REGISTER_CONTEXT(
  )
  RETURN INTEGER;
  ```

- DBE_SQL.SQL_UNREGISTER_CONTEXT

  This function closes a cursor, which is the end of each **dbe_sql** operation. If this function is not called when the stored procedure ends, the memory is still occupied by the cursor. Therefore, remember to close a cursor when you do not need to use it. If an exception occurs, the stored procedure exits but the cursor is not closed. Therefore, you are advised to include this API in the exception handling of the stored procedure.

  The prototype of the DBE_SQL.SQL_UNREGISTER_CONTEXT function is as follows:

  ```
  DBE_SQL.SQL_UNREGISTER_CONTEXT(
      context_id    IN INTEGER
  )
  RETURN INTEGER;
  ```

  Table 10-312 DBE_SQL.SQL_UNREGISTER_CONTEXT parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be closed

- DBE_SQL.SQL_SET_SQL

  Parses SQL statements or anonymous blocks in a given cursor. The statement parameters can be transferred only through the text type. The length cannot exceed 1 GB.

  The prototype of the **DBE_SQL.SQL_SET_SQL** function is as follows:

  ```
  DBE_SQL.SQL_SET_SQL(
      context_id    IN INTEGER,
      query_string  IN TEXT,
  ```



```
    language_flag IN INTEGER
)
RETURN BOOLEAN;
```

Table 10-313 DBE_SQL.SQL_SET_SQL parameters

Parameter	Type	Input/OutputParameter	CanBeEmpty	Description
context_id	INTEGER	IN	No	ID of the cursor whose query statement is to be parsed
query_string	TEXT	IN	No	Query statement to be executed for parsing
language_flag	INTEGER	IN	No	Version language. The value **1** indicates an incompatible version, and the value **2** indicates a compatible version.

- DBE_SQL.SQL_RUN

  This function executes a given cursor. This function receives a cursor ID and executes SQL statements or anonymous blocks in a given cursor.

  The prototype of the **DBE_SQL.SQL_RUN** function is as follows:

```
DBE_SQL.SQL_RUN(
    context_id     IN INTEGER,
)
RETURN INTEGER;
```

Table 10-314 DBE_SQL.SQL_RUN parameters

Parameter	Type	Input/OutputParameter	CanBeEmpty	Description
context_id	INTEGER	IN	No	ID of the cursor whose query statement is to be parsed

- DBE_SQL.NEXT_ROW

  This function returns the number of data rows that meet query conditions. Each time the API is executed, the system obtains a set of new rows until all data is read.

  The prototype of the DBE_SQL.NEXT_ROW function is as follows:

```
DBE_SQL.NEXT_ROW(
    context_id    IN INTEGER,
)
RETURN INTEGER;
```

  Table 10-315 DBE_SQL.NEXT_ROW parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be executed

- DBE_SQL.SET_RESULT_TYPE

  This function defines columns returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

  The prototype of the **DBE_SQL.SET_RESULT_TYPE** function is as follows:

```
DBE_SQL.SET_RESULT_TYPE(
    context_id IN INTEGER,
    pos        IN INTEGER,
    column_ref IN ANYELEMENT,
    maxsize    IN INTEGER default 1024
)
RETURN INTEGER;
```


Table 10-316 DBE_SQL.SET_RESULT_TYPE parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be executed
pos	INTEGER	IN	No	Relative position of the queried columns in the returned result. The value starts from 1.
column_ref	ANYELEMENT	IN	No	Variable of any type. You can select an appropriate API to dynamically define columns based on variable types.
maxsize	INTEGER	IN	Yes	Length of the defined column return type.

- DBE_SQL.SET_RESULT_TYPE_CHAR

  This function defines columns of the **CHAR** type returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

  The prototype of the **DBE_SQL.SET_RESULT_TYPE_CHAR** function is as follows:

```
DBE_SQL.SET_RESULT_TYPE_CHAR(
    context_id  IN INTEGER,
    pos         IN INTEGER,
    column_ref  IN TEXT,
    column_size IN INTEGER
)
RETURN INTEGER;
```


Table 10-317 DBE_SQL.SET_RESULT_TYPE_CHAR parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be executed
pos	INTEGER	IN	No	Position of a dynamically defined column in the query
column_ref	TEXT	IN	No	Parameter to be defined
column_size	INTEGER	IN	No	Length of a dynamically defined column

- DBE_SQL.SET_RESULT_TYPE_INT

  This function defines columns of the **INT** type returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

  The prototype of the **DBE_SQL.SET_RESULT_TYPE_INT** function is as follows:

```
DBE_SQL.SET_RESULT_TYPE_INT(
    context_id IN INTEGER,
    pos        IN INTEGER
)
RETURN INTEGER;
```


Table 10-318 DBE_SQL.SET_RESULT_TYPE_INT parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be executed
pos	INTEGER	IN	No	Position of a dynamically defined column in the query

- DBE_SQL.SET_RESULT_TYPE_LONG

  This function defines columns of a long type (not **LONG**) returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type. The maximum size of a long column is 1 GB.

  The prototype of the **DBE_SQL.SET_RESULT_TYPE_LONG** function is as follows:

```
DBE_SQL.SET_RESULT_TYPE_LONG(
   context_id IN INTEGER,
   pos        IN INTEGER
)
RETURN INTEGER;
```


Table 10-319 DBE_SQL.SET_RESULT_TYPE_LONG parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be executed
pos	INTEGER	IN	No	Position of a dynamically defined column in the query

- DBE_SQL.SET_RESULT_TYPE_RAW

  This function defines columns of the **RAW** type returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

  The prototype of the **DBE_SQL.SET_RESULT_TYPE_RAW** function is as follows:

```
DBE_SQL.SET_RESULT_TYPE_RAW(
   context_id  IN INTEGER,
   pos         IN INTEGER,
   column_ref  IN RAW,
   column_size IN INTEGER
)
RETURN INTEGER;
```


Table 10-320 DBE_SQL.SET_RESULT_TYPE_RAW parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be executed
pos	INTEGER	IN	No	Position of a dynamically defined column in the query
column_ref	RAW	IN	No	RAW variable
column_size	INTEGER	IN	No	Column length

- DBE_SQL.SET_RESULT_TYPE_BYTEA

  Defines columns of the BYTEA type returned from a given cursor and can be used only for cursors defined by SELECT. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

  The prototype of the DBE_SQL.SET_RESULT_TYPE_BYTEA function is as follows:

```
DBE_SQL.SET_RESULT_TYPE_BYTEA(
   context_id  IN INTEGER,
   pos         IN INTEGER,
   column_ref  IN BYTEA,
   column_size IN INTEGER
)
RETURN INTEGER;
```


Table 10-321 DBE_SQL.SET_RESULT_TYPE_BYTEA parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be executed
pos	INTEGER	IN	No	Position of a dynamically defined column in the query
column_ref	BYTEA	IN	No	BYTEA variable
column_size	INTEGER	IN	No	Column length

- DBE_SQL.SET_RESULT_TYPE_TEXT

  This function defines columns of the **TEXT** type returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

  The prototype of the **DBE_SQL.SET_RESULT_TYPE_TEXT** function is as follows:

```
DBE_SQL.DEFINE_COLUMN_CHAR(
    context_id IN INTEGER,
    pos        IN INTEGER,
    maxsize    IN INTEGER
)
RETURN INTEGER;
```


Table 10-322 DBE_SQL.SET_RESULT_TYPE_TEXT parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be executed
pos	INTEGER	IN	No	Position of a dynamically defined column in the query
maxsize	INTEGER	IN	No	Maximum length of the defined TEXT type

- DBE_SQL.SET_RESULT_TYPE_UNKNOWN

  This function processes columns of unknown data types returned from a given cursor. It is used only for the system to report an error and exist when the type cannot be identified.

  The prototype of the **DBE_SQL.SET_RESULT_TYPE_UNKNOWN** function is as follows:

```
DBE_SQL.SET_RESULT_TYPE_UNKNOWN(
    context_id IN INTEGER,
    pos        IN INTEGER,
    col_type   IN TEXT
)
RETURN INTEGER;
```

Table 10-323 DBE_SQL.SET_RESULT_TYPE_UNKNOWN parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be executed

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
pos	INTEGER	IN	No	Position of a dynamically defined column in the query
col_type	TEXT	IN	No	Dynamically defined parameter

- DBE_SQL.GET_RESULT

  This function returns the cursor element value in a specified position of a cursor and accesses the data obtained by **DBE_SQL.NEXT_ROW**.

  The prototype of the **DBE_SQL.GET_RESULT** function is as follows:

```
DBE_SQL.GET_RESULT(
   context_id   IN    INTEGER,
   pos          IN    INTEGER,
   column_value INOUT ANYELEMENT
)
RETURN ANYELEMENT;
```

  Table 10-324 DBE_SQL.GET_RESULT parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be executed
pos	INTEGER	IN	No	Relative position of the queried columns in the returned result. The value starts from 1.

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
column_value	ANYELEMENT	INOUT	No	Return value of the query result of a specified column

- DBE_SQL.GET_RESULT_CHAR

  This stored procedure returns the value of the **CHAR** type in a specified position of a cursor and accesses the data obtained by **DBE_SQL.NEXT_ROW**.

  The prototype of the DBE_SQL.GET_RESULT_CHAR stored procedure is as follows:

```
DBE_SQL.GET_RESULT_CHAR(
    context_id    IN    INTEGER,
    pos           IN    INTEGER,
    tr            INOUT CHAR,
    err           INOUT NUMERIC,
    actual_length INOUT INTEGER
);
```

Table 10-325 DBE_SQL.GET_RESULT_CHAR parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be executed

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
pos	INTEGER	IN	No	Relative position of the queried columns in the returned result. The value starts from 1.
tr	CHAR	INOUT	No	Return value
err	NUMERIC	INOUT	No	Error No. It is an output parameter. The input parameter must be a variable. Currently, the output value is **–1** regardless of the input parameter.
actual_length	INTEGER	INOUT	No	Length of a return value

- The overloading of the DBE_SQL.GET_RESULT_CHAR stored procedure is as follows:

```
DBE_SQL.GET_RESULT_CHAR(
   context_id IN    INTEGER,
   pos        IN    INTEGER,
   tr         INOUT CHAR
);
```


Table 10-326 DBE_SQL.GET_RESULT_CHAR parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be executed
pos	INTEGER	IN	No	Position of a dynamically defined column in the query
tr	CHAR	INOUT	No	Return value

- DBE_SQL.GET_RESULT_INT

  This function returns the value of the **INT** type in a specified position of a cursor and accesses the data obtained by **DBE_SQL.NEXT_ROW**. The prototype of the **DBE_SQL.GET_RESULT_INT** function is as follows:

```
DBE_SQL.GET_RESULT_INT(
    context_id IN INTEGER,
    pos        IN INTEGER
)
RETURN INTEGER;
```


Table 10-327 DBE_SQL.GET_RESULT_INT parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be executed
pos	INTEGER	IN	No	Position of a dynamically defined column in the query

- DBE_SQL.GET_RESULT_LONG

  This function returns the value of a long type (not **LONG** or **BIGINT**) in a specified position of a cursor and accesses the data obtained by **DBE_SQL.NEXT_ROW**.

  The prototype of the **DBE_SQL.GET_RESULT_LONG** function is as follows:

```
DBE_SQL.GET_RESULT_LONG(
    context_id IN INTEGER,
    pos        IN    INTEGER,
    lgth       IN    INTEGER,
    off_set    IN    INTEGER,
    vl         INOUT TEXT,
    vl_length  INOUT INTEGER
)
RETURN RECORD;
```


Table 10-328 DBE_SQL.GET_RESULT_LONG parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be executed
pos	INTEGER	IN	No	Position of a dynamically defined column in the query
lgth	INTEGER	IN	No	Length of a return value
off_set	INTEGER	IN	No	Start position of a return value
vl	TEXT	INOUT	No	Return value
vl_length	INTEGER	INOUT	No	Length of a return value

- DBE_SQL.GET_RESULT_RAW

  Returns the value of the RAW type in a specified position of a cursor and accesses the data obtained by DBE_SQL.NEXT_ROW.

  The prototype of the DBE_SQL.GET_RESULT_RAW stored procedure is as follows:

```
DBE_SQL.GET_RESULT_RAW(
    context_id    IN    INTEGER,
    pos           IN    INTEGER,
    tr            INOUT RAW,
    err           INOUT NUMERIC,
    actual_length INOUT INTEGER
);
```


Table 10-329 DBE_SQL.GET_RESULT_RAW parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be executed
pos	INTEGER	IN	No	Position of a dynamically defined column in the query
tr	RAW	INOUT	No	Returned column value
err	NUMERIC	INOUT	No	Error No. It is an output parameter. The input parameter must be a variable. Currently, the output value is **–1** regardless of the input parameter.
actual_length	INTEGER	INOUT	No	Length of a return value. The value longer than this length will be truncated.

- The overloading of the DBE_SQL.GET_RESULT_RAW stored procedure is as follows:

```
DBE_SQL.GET_RESULT_RAW(
    context_id IN     INTEGER,
    pos        IN     INTEGER,
    tr         INOUT RAW
);
```


Table 10-330 DBE_SQL.GET_RESULT_RAW parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be executed
pos	INTEGER	IN	No	Position of a dynamically defined column in the query
tr	RAW	INOUT	No	Returned column value

- DBE_SQL.GET_RESULT_BYTEA

  Returns the value of the BYTEA type in a specified position of a cursor that is obtained by DBE_SQL.NEXT_ROW.

  The prototype of the DBE_SQL.GET_RESULT_BYTEA stored procedure is as follows:

```
DBE_SQL.GET_RESULT_BYTEA(
   context_id IN INTEGER,
   pos        IN INTEGER
)
RETURN BYTEA;
```


Table 10-331 DBE_SQL.GET_RESULT_BYTEA parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be executed
pos	INTEGER	IN	No	Position of a dynamically defined column in the query

- DBE_SQL.GET_RESULT_TEXT

  This function returns the value of the **TEXT** type in a specified position of a cursor and accesses the data obtained by **DBE_SQL.NEXT_ROW**.

  The prototype of the **DBE_SQL.GET_RESULT_TEXT** function is as follows:

```
DBE_SQL.GET_RESULT_TEXT(
    context_id IN  INTEGER,
    pos        IN  INTEGER
)
RETURN TEXT;
```

Table 10-332 DBE_SQL.GET_RESULT_TEXT parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be executed

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
pos	INTEGER	IN	No	Position of a dynamically defined column in the query

- DBE_SQL.GET_RESULT_UNKNOWN

  This function returns the value of an unknown type in a specified position of a cursor. It serves as an error handling API when the type is not unknown.

  The prototype of the DBE_SQL.GET_RESULT_UNKNOWN function is as follows:

```
DBE_SQL.GET_RESULT_UNKNOWN(
    context_id IN INTEGER,
    pos        IN INTEGER,
    col_type   IN TEXT
)
RETURN TEXT;
```

Table 10-333 DBE_SQL.GET_RESULT_UNKNOWN parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be executed
pos	INTEGER	IN	No	Position of a dynamically defined column in the query

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
col_type	TEXT	IN	No	Returned parameter type

- DBE_SQL.DBE_SQL_GET_RESULT_CHAR

  This function returns the value of the **CHAR** type in a specified position of a cursor and accesses the data obtained by **DBE_SQL.NEXT_ROW**. Different from **DBE_SQL.GET_RESULT_CHAR**, the length of the return value is not set and the entire string is returned.

  The prototype of the **DBE_SQL.DBE_SQL_GET_RESULT_CHAR** function is as follows:

```
DBE_SQL.DBE_SQL_GET_RESULT_CHAR(
    context_id IN INTEGER,
    pos        IN INTEGER
)
RETURN CHARACTER;
```

  Table 10-334 DBE_SQL.DBE_SQL_GET_RESULT_CHAR parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be executed
pos	INTEGER	IN	No	Position of a dynamically defined column in the query

- DBE_SQL.DBE_SQL_GET_RESULT_LONG

  This function returns the value of a long type (not **LONG** or **BIGINT**) in a specified position of a cursor and accesses the data obtained by **DBE_SQL.NEXT_ROW**.

Different from **DBE_SQL.GET_RESULT_LONG**, the length of the return value is not set and the entire BIGINT value is returned.

The prototype of the **DBE_SQL.DBE_SQL_GET_RESULT_LONG** function is as follows:

```
DBE_SQL.DBE_SQL_GET_RESULT_LONG(
    context_id IN INTEGER,
    pos        IN INTEGER
)
RETURN BIGINT;
```

Table 10-335 DBE_SQL.DBE_SQL_GET_RESULT_LONG parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be executed
pos	INTEGER	IN	No	Position of a dynamically defined column in the query

- DBE_SQL.DBE_SQL_GET_RESULT_RAW

  This function returns the value of the **RAW** type in a specified position of a cursor and accesses the data obtained by **DBE_SQL.NEXT_ROW**.

  Different from **DBE_SQL.GET_RESULT_RAW**, the length of the return value is not set and the entire string is returned.

  The prototype of the **DBE_SQL.DBE_SQL_GET_RESULT_RAW** function is as follows:

  ```
  DBE_SQL.GET_RESULT_RAW(
      context_id IN    INTEGER,
      pos        IN    INTEGER,
      tr         INOUT RAW
  )
  RETURN RAW;
  ```


Table 10-336 DBE_SQL.GET_RESULT_RAW parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be executed
pos	INTEGER	IN	No	Position of a dynamically defined column in the query
tr	RAW	INOUT	No	Returned column value

- DBE_SQL.IS_ACTIVE

  This function returns the status of a cursor. When a cursor is opened, parsed, executed, or defined, the value is **TRUE**. When a cursor is closed, the value is **FALSE**. If the status is unknown, an error is reported. In other cases, the cursor is closed by default.

  The prototype of the **DBE_SQL.IS_ACTIVE** function is as follows:

```
DBE_SQL.IS_ACTIVE(
   context_id     IN   INTEGER
)
RETURN BOOLEAN;
```


Table 10-337 DBE_SQL.IS_ACTIVE parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor to be queried

- DBE_SQL.LAST_ROW_COUNT

  Returns the accumulated count of data rows obtained after the latest NEXT_ROW execution.

  The prototype of the DBE_SQL.LAST_ROW_COUNT function is as follows:

  ```
  DBE_SQL.LAST_ROW_COUNT(
  )
  RETURN INTEGER;
  ```

- DBE_SQL.RUN_AND_NEXT

  Equivalent to calling SQL_RUN and NEXT_ROW in sequence.

  The prototype of the DBE_SQL.RUN_AND_NEXT function is as follows:

  ```
  DBE_SQL.RUN_AND_NEXT(
     context_id IN INTEGER
  )
  RETURNS INTEGER;
  ```

Table 10-338 DBE_SQL.RUN_AND_NEXT parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INTEGER	IN	No	ID of the cursor whose query statement is to be parsed

- DBE_SQL.SQL_BIND_VARIABLE

  This function is used to bind a parameter to an SQL statement. When an SQL statement is executed, the SQL statement is executed based on the bound value.

  The prototype of the **DBE_SQL.SQL_BIND_VARIABLE** function is as follows:

```
DBE_SQL.SQL_BIND_VARIABLE(
    context_id      IN int,
    query_string    IN text,
    language_flag   IN anyelement,
    out_value_size IN int default null
)
RETURNS void;
```

  Table 10-339 DBE_SQL.SQL_BIND_VARIABLE parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
query_string	TEXT	IN	No	Name of the bound variable
language_flag	ANYELEMENT	IN	No	Bound value
out_value_size	INT	IN	Yes	Size of the return value. Default value: **NULL**. In A-compatible mode, the parameter behavior is the same as that of database A only when the parameter type of the value is VARCHAR or CHAR.

- DBE_SQL.SQL_BIND_ARRAY

  This function is used to bind a set of parameters to an SQL statement. When an SQL statement is executed, the SQL statement is executed based on the bound array.

The prototype of the **DBE_SQL.SQL_BIND_ARRAY** function is as follows:

```
DBE_SQL.SQL_BIND_ARRAY(
    context_id   IN int,
    query_string IN text,
    value        IN anyarray
)
RETURNS void;
DBE_SQL.SQL_BIND_ARRAY(
    context_id   IN int,
    query_string IN text,
    value        IN anyarray,
    lower_index  IN int,
    higher_index IN int
)
RETURNS void;

DBE_SQL.SQL_BIND_ARRAY(
    context_id   IN int,
    query_string IN text,
    value        IN anyindexbytable
)
RETURNS void;
DBE_SQL.SQL_BIND_ARRAY(
    context_id   IN int,
    query_string IN text,
    value        IN anyindexbytable,
    lower_index  IN int,
    higher_index IN int
)
RETURNS void;
```

Table 10-340 DBE_SQL.SQL_BIND_ARRAY parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried.
query_string	TEXT	IN	No	Name of the bound variable.
value	ANYINDEXBYTABLE	IN	No	Bound array.
lower_index	INT	IN	No	Minimum index of the bound array.
higher_index	INT	IN	No	Maximum index of the bound array.

> **NOTE**
>
> DBE_SQL.SQL_BIND_ARRAY does not support user-defined table types. Use the table types provided in **Data Types**.

- DBE_SQL.SET_RESULT_TYPE_INTS

  This function defines columns of the **INT** array type returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

  The prototype of the **DBE_SQL.SET_RESULT_TYPE_INTS** function is as follows:

```
DBE_SQL.SET_RESULT_TYPE_INTS(
    context_id IN int,
    pos        IN int,
    column_ref IN anyarray,
    cnt        IN int,
    lower_bnd  IN int
)
RETURNS integer;
```

Table 10-341 DBE_SQL.SET_RESULT_TYPE_INTS parameters

Parameter	Type	Input/OutputParameter	CanBeEmpty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	INT	IN	No	Position of a dynamically defined column in the query
column_ref	ANYARRAY	IN	No	Type of the returned array
cnt	INT	IN	No	Number of values obtained at a time
lower_bnd	INT	IN	No	Start index when an array is returned.

- DBE_SQL.SET_RESULT_TYPE_TEXTS

  This function defines columns of the TEXT array type returned from a given cursor and can be used only for the cursors defined by SELECT. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

The prototype of the DBE_SQL.SET_RESULT_TYPE_TEXTS function is as follows:

```
DBE_SQL.SET_RESULT_TYPE_TEXTS(
    context_id IN int,
    pos        IN int,
    column_ref IN anyarray,
    cnt        IN int,
    lower_bnd  IN int,
    maxsize    IN int
)
RETURNS integer;
```

Table 10-342 DBE_SQL.SET_RESULT_TYPE_TEXTS parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	INT	IN	No	Position of a dynamically defined column in the query
column_ref	ANYARRAY	IN	No	Type of the returned array
cnt	INT	IN	No	Number of values obtained at a time
lower_bnd	INT	IN	No	Start index when an array is returned.
maxsize	INT	IN	No	Maximum length of the defined TEXT type

- DBE_SQL.SET_RESULT_TYPE_RAWS

  This function defines columns of the **RAW** array type returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

  The prototype of the **DBE_SQL.SET_RESULT_TYPE_RAWS** function is as follows:

```
DBE_SQL.set_result_type_raws(
    context_id  IN int,
    pos         IN int,
    column_ref  IN anyarray,
    cnt         IN int,
    lower_bnd   IN int,
```



```
    column_size IN int
)
RETURNS integer;
```

Table 10-343 DBE_SQL.SET_RESULT_TYPE_RAWS parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	INT	IN	No	Position of a dynamically defined column in the query
column_ref	ANYARRAY	IN	No	Type of the returned array
cnt	INT	IN	No	Number of values obtained at a time
lower_bnd	INT	IN	No	Start index when an array is returned.
column_size	INT	IN	No	Column length

- DBE_SQL.SET_RESULT_TYPE_BYTEAS

  This function defines columns of the **BYTEA** array type returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

  The prototype of the **DBE_SQL.SET_RESULT_TYPE_BYTEAS** function is as follows:

```
DBE_SQL.set_result_type_byteas(
    context_id  IN int,
    pos         IN int,
    column_ref  IN anyarray,
    cnt         IN int,
    lower_bnd   IN int,
    column_size IN int
)
RETURNS integer;
```


Table 10-344 DBE_SQL.SET_RESULT_TYPE_BYTEAS parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	INT	IN	No	Position of a dynamically defined column in the query
column_ref	ANYARRAY	IN	No	Type of the returned array
cnt	INT	IN	No	Number of values obtained at a time
lower_bnd	INT	IN	No	Start index when an array is returned.
column_size	INT	IN	No	Column length

- DBE_SQL.SET_RESULT_TYPE_CHARS

  This function defines columns of the **CHAR** array type returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

  The prototype of the **DBE_SQL.SET_RESULT_TYPE_CHARS** function is as follows:

```
DBE_SQL.SET_RESULT_TYPE_CHARS(
    context_id  IN int,
    pos         IN int,
    column_ref  IN anyarray,
    cnt         IN int,
    lower_bnd   IN int,
    column_size IN int
)
RETURNS integer;
```


Table 10-345 DBE_SQL.SET_RESULT_TYPE_CHARS parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	INT	IN	No	Position of a dynamically defined column in the query
column_ref	ANYARRAY	IN	No	Type of the returned array
cnt	INT	IN	No	Number of values obtained at a time
lower_bnd	INT	IN	No	Start index when an array is returned.
column_size	INT	IN	No	Column length

- DBE_SQL.SET_RESULTS_TYPE

  This function defines columns returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of an input variable determines the corresponding column type.

  The prototype of the **DBE_SQL.SET_RESULTS_TYPE** function is as follows:

```
DBE_SQL.SET_RESULTS_TYPE(
    context_id IN int,
    pos        IN int,
    column_ref IN anyarray,
    cnt        IN int,
    lower_bnd  IN int,
    maxsize    IN int DEFAULT 1024
) returns void;

DBE_SQL.SET_RESULTS_TYPE(
    context_id IN int,
    pos        IN int,
    column_ref IN dbe_sql.number_table,
    cnt        IN int,
    lower_bnd  IN int,
    maxsize    IN int DEFAULT 1024
);

DBE_SQL.SET_RESULTS_TYPE(
    context_id IN int,
    pos        IN int,
    column_ref IN dbe_sql.varchar2_table,
    cnt        IN int,
    lower_bnd  IN int,
    maxsize    IN int DEFAULT 32767
```



```
);

DBE_SQL.SET_RESULTS_TYPE(
    context_id IN int,
    pos        IN int,
    column_ref IN dbe_sql.date_table,
    cnt        IN int,
    lower_bnd  IN int,
    maxsize    IN int DEFAULT 1024
);

DBE_SQL.SET_RESULTS_TYPE(
    context_id IN int,
    pos        IN int,
    column_ref IN dbe_sql.blob_table,
    cnt        IN int,
    lower_bnd  IN int,
    maxsize    IN int DEFAULT 32767
);
```

Table 10-346 DBE_SQL.SET_RESULTS_TYPE parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	INT	IN	No	Position of a dynamically defined column in the query
column_ref	DBE_SQL.BLOB_TABLE	IN	No	Type of the returned array
cnt	INT	IN	No	Number of values obtained at a time
lower_bnd	INT	IN	No	Start index when an array is returned.
maxsize	INT	IN	Yes	Maximum length of the defined type

NOTE

DBE_SQL.SET_RESULTS_TYPE does not support user-defined table types. Use the table types provided in "Data Types".

- DBE_SQL.GET_RESULTS_INT

  This function returns the value of the **INT** array type in a specified position of a cursor and accesses the data obtained by **DBE_SQL.NEXT_ROW**.

The prototype of the **DBE_SQL.GET_RESULTS_INT** function is as follows:

```
DBE_SQL.GET_RESULTS_INT(
   context_id   IN    int,
   pos          IN    int,
   column_value INOUT anyarray
);
```

Table 10-347 DBE_SQL.GET_RESULTS_INT parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	INT	IN	No	Position of a dynamically defined column in the query
column_value	ANYARRAY	INOUT	No	Return value

- DBE_SQL.GET_RESULTS_TEXT

  This function returns the value of the **TEXT** array type in a specified position of a cursor and accesses the data obtained by **DBE_SQL.NEXT_ROW**.

  The prototype of the **DBE_SQL.GET_RESULTS_TEXT** function is as follows:

```
DBE_SQL.GET_RESULTS_TEXT(
   context_id   IN    int,
   pos          IN    int,
   column_value INOUT anyarray
);
```

Table 10-348 DBE_SQL.GET_RESULTS_TEXT parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
pos	INT	IN	No	Position of a dynamically defined column in the query
column_value	ANYARRAY	INOUT	No	Return value

- DBE_SQL.GET_RESULTS_RAW

  This function returns the value of the **RAW** array type in a specified position of a cursor and accesses the data obtained by **DBE_SQL.NEXT_ROW**.

  The prototype of the **DBE_SQL.GET_RESULTS_RAW** function is as follows:

```
DBE_SQL.GET_RESULTS_RAW(
   context_id   IN    int,
   pos          IN    int,
   column_value INOUT anyarray
);
```

Table 10-349 DBE_SQL.GET_RESULTS_RAW parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	INT	IN	No	Position of a dynamically defined column in the query
column_value	ANYARRAY	INOUT	No	Return value

- DBE_SQL.GET_RESULTS_BYTEA

  This function returns the value of the **BYTEA** array type in a specified position of a cursor and accesses the data obtained by **DBE_SQL.NEXT_ROW**.

  The prototype of the **DBE_SQL.GET_RESULTS_BYTEA** function is as follows:

```
DBE_SQL.GET_RESULTS_BYTEA(
   context_id   IN    int,
   pos          IN    int,
   column_value INOUT anyarray
);
```

  Table 10-350 DBE_SQL.GET_RESULTS_BYTEA parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	INT	IN	No	Position of a dynamically defined column in the query
column_value	ANYARRAY	INOUT	No	Return value

- DBE_SQL.GET_RESULTS_CHAR

  This function returns the value of the **CHAR** array type in a specified position of a cursor and accesses the data obtained by **DBE_SQL.NEXT_ROW**.

  The prototype of the **DBE_SQL.GET_RESULTS_CHAR** function is as follows:

```
DBE_SQL.GET_RESULTS_CHAR(
   context_id   IN    int,
   pos          IN    int,
   column_value INOUT anyarray
);
```


Table 10-351 DBE_SQL.GET_RESULTS_CHAR parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	INT	IN	No	Position of a dynamically defined column in the query
column_value	ANYARRAY	INOUT	No	Return value

- DBE_SQL.GET_RESULTS

  This function returns the value of the array type in a specified position of a cursor and accesses the data obtained by **DBE_SQL.NEXT_ROW**.

  NOTE

  The bottom-layer mechanism of DBE_SQL.GET_RESULTS is implemented through arrays. When different arrays are used to obtain the return value of the same column, NULL values are filled in the array due to discontinuous internal indexes to ensure the continuity of array indexes. As a result, the length of the returned result array is different from that of the Oracle database.

  The prototype of the **DBE_SQL.GET_RESULTS** function is as follows:

```
DBE_SQL.GET_RESULTS(
   context_id   IN    int,
   pos          IN    int,
   column_value INOUT anyarray
);

DBE_SQL.GET_RESULTS(
   context_id   IN    int,
   pos          IN    int,
   column_value INOUT dbe_sql.varchar2_table
);

DBE_SQL.GET_RESULTS(
   context_id   IN    int,
   pos          IN    int,
   column_value INOUT dbe_sql.number_table
);

DBE_SQL.GET_RESULTS(
   context_id   IN    int,
   pos          IN    int,
   column_value INOUT dbe_sql.date_table
);

DBE_SQL.GET_RESULTS(
```



```
  context_id   IN    int,
  pos          IN    int,
  column_value INOUT dbe_sql.blob_table
);
```

Table 10-352 DBE_SQL.GET_RESULTS parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	INT	IN	No	Position of a dynamically defined column in the query
column_value	ANYARRAY	INOUT	No	Return value

NOTE

DBE_SQL.GET_RESULTS does not support user-defined table types. Use the table types provided in **Data Types**.

- DBE_SQL.SQL_DESCRIBE_COLUMNS

  This function is used to describe column information and can be used only for cursors defined by **SELECT**.

  The prototype of the **DBE_SQL.SQL_DESCRIBE_COLUMNS** function is as follows:

```
DBE_SQL.SQL_DESCRIBE_COLUMNS(
  context_id IN    int,
  col_cnt    INOUT int,
  desc_t     INOUT dbe_sql.desc_tab
)RETURNS record ;
```


Table 10-353 DBE_SQL.SQL_DESCRIBE_COLUMNS parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
col_cnt	INT	INOUT	No	Number of columns returned
desc_t	DBE_SQL.DESC_TAB	INOUT	No	Description of the returned column

- DBE_SQL.DESCRIBE_COLUMNS

  Describes column information and can be used only for cursors defined by SELECT. This API is developed for compatibility purposes.

  The prototype of the DBE_SQL.DESCRIBE_COLUMNS function is as follows:

```
DBE_SQL.DESCRIBE_COLUMNS(
    context_id IN  int,
    col_cnt    OUT int,
    desc_t     OUT dbe_sql.desc_tab
)
```

Table 10-354 DBE_SQL.DESCRIBE_COLUMNS parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
col_cnt	INT	OUT	No	Number of columns returned

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
desc_t	DBE_SQL.DESC_TAB	OUT	No	Description of the returned column

- DBE_SQL.BIND_VARIABLE

  This function is used to bind parameters. You are advised to use **DBE_SQL.SQL_BIND_VARIABLE**.

- DBE_SQL.SQL_SET_RESULTS_TYPE_C

  This function is used to dynamically define a column of the array type. You are advised not to use it.

  The prototype of the **DBE_SQL.SQL_SET_RESULTS_TYPE_C** function is as follows:

```
DBE_SQL.sql_set_results_type_c(
   context_id IN int,
   pos        IN int,
   column_ref IN anyarray,
   cnt        IN int,
   lower_bnd  IN int,
   col_type   IN anyelement,
   maxsize    IN int
)return integer;
```

Table 10-355 DBE_SQL.SQL_SET_RESULTS_TYPE_C parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	INT	IN	No	Position of a dynamically defined column in the query

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
column_ref	ANYARRAY	IN	No	Type of the returned array
cnt	INT	IN	No	Number of values obtained at a time
lower_bnd	INT	IN	No	Start index when an array is returned.
col_type	ANYELEMENT	IN	No	Variable type corresponding to the returned array type
maxsize	INT	IN	No	Maximum length of the defined type

- DBE_SQL.SQL_GET_VALUES_C

  This function is used to read a dynamically defined column value. You are advised not to use it.

  The prototype of the **DBE_SQL.SQL_GET_VALUES_C** function is as follows:

```
DBE_SQL.sql_get_values_c(
    context_id   IN    int,
    pos          IN    int,
    results_type INOUT anyarray,
    result_type  IN    anyelement
)return anyarray;
```

Table 10-356 DBE_SQL.SQL_GET_VALUES_C parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	INT	IN	No	Parameter position

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
results_type	ANY ARRAY	IN OUT	No	Obtained result
result_type	ANY ELEMENT	IN	No	Type of the obtained result

- DBE_SQL.GET_VARIABLE_RESULT

  This function is used to return the value of the bound OUT parameter and obtain the OUT parameter in a stored procedure.

  The prototype of the **DBE_SQL.GET_VARIABLE_RESULT** function is as follows:

```
DBE_SQL.get_variable_result(
    context_id   IN    int,
    pos          IN    VARCHAR2,
    column_value INOUT anyelement
);
```

Table 10-357 DBE_SQL.GET_VARIABLE_RESULT parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	VARCHAR2	IN	No	Name of the bound parameter

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
column_value	ANYELEMENT	INOUT	No	Return value

- DBE_SQL.GET_VARIABLE_RESULT_CHAR

  This function is used to return the value of the bound OUT parameter of the **CHAR** type and obtain the OUT parameter in a stored procedure.

  The prototype of the **DBE_SQL.GET_VARIABLE_RESULT_CHAR** function is as follows:

```
DBE_SQL.get_variable_result_char(
    context_id IN int,
    pos        IN VARCHAR2
)
RETURNS char
```

  Table 10-358 DBE_SQL.GET_VARIABLE_RESULT_CHAR parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	VARCHAR2	IN	No	Name of the bound parameter

- DBE_SQL.GET_VARIABLE_RESULT_RAW

  This function is used to return the value of the bound OUT parameter of the **RAW** type and obtain the OUT parameter in a stored procedure.

The prototype of the **DBE_SQL.GET_VARIABLE_RESULT_RAW** function is as follows:

```
CREATE OR REPLACE FUNCTION DBE_SQL.get_variable_result_raw(
   context_id IN    int,
   pos       IN    VARCHAR2,
   value     INOUT anyelement
)
RETURNS anyelement
```

Table 10-359 DBE_SQL.GET_VARIABLE_RESULT_RAW parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	VARCHAR2	IN	No	Name of the bound parameter
value	ANYELEMENT	INOUT	No	Return value

- DBE_SQL.GET_VARIABLE_RESULT_TEXT

  This function is used to return the value of the bound OUT parameter of the **TEXT** type and obtain the OUT parameter in a stored procedure.

  The prototype of the **DBE_SQL.GET_VARIABLE_RESULT_TEXT** function is as follows:

  ```
  CREATE OR REPLACE FUNCTION DBE_SQL.get_variable_result_text(
     context_id IN int,
     pos       IN VARCHAR2
  )
  RETURNS text
  ```


Table 10-360 DBE_SQL.GET_VARIABLE_RESULT_TEXT parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	VARCHAR2	IN	No	Name of the bound parameter

- DBE_SQL.GET_VARIABLE_RESULT_INT

  This function is used to return the value of the bound OUT parameter of the **INT** type and obtain the OUT parameter in a stored procedure.

  The prototype of the **DBE_SQL.GET_VARIABLE_RESULT_INT** function is as follows:

```
DBE_SQL.get_variable_result_int(
    context_id IN     int,
    pos        IN     VARCHAR2,
    value      INOUT anyelement
)
RETURNS anyelement
```

Table 10-361 DBE_SQL.GET_VARIABLE_RESULT_INT parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	VARCHAR2	IN	No	Name of the bound parameter
value	ANYELEMENT	INOUT	No	Return value

- DBE_SQL.GET_ARRAY_RESULT_TEXT

  This function is used to return the value of the bound OUT parameter of the **TEXT** array type and obtain the OUT parameter in a stored procedure.

  The prototype of the **DBE_SQL.GET_ARRAY_RESULT_TEXT** function is as follows:

```
DBE_SQL.get_array_result_text(
    context_id   IN    int,
    pos          IN    VARCHAR2,
    column_value INOUT anyarray
)
```

  Table 10-362 DBE_SQL.GET_ARRAY_RESULT_TEXT parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	VARCHAR2	IN	No	Name of the bound parameter
column_value	ANYARRAY	INOUT	No	Return value

- DBE_SQL.GET_ARRAY_RESULT_RAW

  This function is used to return the value of the bound OUT parameter of the **RAW** array type and obtain the OUT parameter in a stored procedure.

  The prototype of the **DBE_SQL.GET_ARRAY_RESULT_RAW** function is as follows:

```
DBE_SQL.get_array_result_raw(
    context_id   IN    int,
    pos          IN    VARCHAR2,
    column_value INOUT anyarray
)
```


Table 10-363 DBE_SQL.GET_ARRAY_RESULT_RAW parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	VARCHAR2	IN	No	Name of the bound parameter
column_value	ANYARRAY	INOUT	No	Return value

- DBE_SQL.GET_ARRAY_RESULT_CHAR

  This function is used to return the value of the bound OUT parameter of the **CHAR** array type and obtain the OUT parameter in a stored procedure.

  The prototype of the **DBE_SQL.GET_ARRAY_RESULT_CHAR** function is as follows:

```
DBE_SQL.get_array_result_char(
    context_id   IN    int,
    pos          IN    VARCHAR2,
    column_value INOUT anyarray
)
```

Table 10-364 DBE_SQL.GET_ARRAY_RESULT_CHAR parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	VARCHAR2	IN	No	Name of the bound parameter

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
column_value	ANY ARRAY	INOUT	No	Return value

- DBE_SQL.GET_ARRAY_RESULT_INT

  This function is used to return the value of the bound OUT parameter of the **INT** array type and obtain the OUT parameter in a stored procedure.

  The prototype of the **DBE_SQL.GET_ARRAY_RESULT_INT** function is as follows:

```
DBE_SQL.get_array_result_int(
    context_id   IN    int,
    pos          IN    VARCHAR2,
    column_value INOUT anyarray
)
```

  Table 10-365 DBE_SQL.GET_ARRAY_RESULT_INT parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	VARCHAR2	IN	No	Name of the bound parameter
column_value	ANYARRAY	INOUT	No	Return value

- DBE_SQL.SQL_SET_TABLEOF_RESULTS_TYPE_C

  Dynamically defines a column of the tableof type. You are advised not to use it.

The prototype of the DBE_SQL.SQL_SET_TABLEOF_RESULTS_TYPE_C function is as follows:

```
DBE_SQL.SQL_SET_TABLEOF_RESULTS_TYPE_C(
    context_id IN int,
    pos        IN int,
    column_ref IN anyindexbytable,
    cnt        IN int,
    lower_bnd  IN int,
    col_type   IN anyelement,
    maxsize    IN int
)return integer;
```

Table 10-366 DBE_SQL.SQL_SET_TABLEOF_RESULTS_TYPE_C parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	INT	IN	No	Position of a dynamically defined column in the query
column_ref	ANYINDEXBYTABLE	IN	No	Type of the returned array
cnt	INT	IN	No	Number of values obtained at a time
lower_bnd	INT	IN	No	Start index when an array is returned.
col_type	ANYELEMENT	IN	No	Variable type corresponding to the returned array type
maxsize	INT	IN	No	Maximum length of the defined type

- DBE_SQL.SQL_GET_TABLEOF_VALUES_C

  Reads a dynamically defined column value of the tableof type. You are advised not to use it.

  The function prototype of DBE_SQL.SQL_GET_TABLEOF_VALUES_C is as follows:

```
DBE_SQL.SQL_GET_TABLEOF_VALUES_C(
    context_id   IN    int,
    pos          IN    int,
    results_type INOUT anyindexbytable,
    result_type  IN    anyelement
)return anyindexbytable;
```


Table 10-367 DBE_SQL.SQL_GET_TABLEOF_VALUES_C parameters

Parameter	Type	Input/OutputParameter	Can Be Empty	Description
context_id	INT	IN	No	ID of the cursor to be queried
pos	INT	IN	No	Parameter position
results_type	ANYINDEXBYTABLE	INOUT	No	Obtained result
result_type	ANYELEMENT	IN	No	Type of the obtained result

Examples

```
-- Example 1
-- Create a table and insert data into the table.
CREATE TABLE test_desc_cols(
    id NUMBER,
    name VARCHAR2(50)
);
INSERT INTO test_desc_cols(id, name) VALUES (1, 'xiaoming');
INSERT INTO test_desc_cols(id, name) VALUES (2, 'xiaohong');
INSERT INTO test_desc_cols(id, name) VALUES (3, 'xiaolan');

DECLARE
context_id  INTEGER;
col_cnt     INTEGER;
v_id int;
v_name varchar2;
execute_ret  INTEGER;
BEGIN
-- Open a cursor.
context_id := DBE_SQL.REGISTER_CONTEXT();
-- Compile the cursor.
DBE_SQL.SQL_SET_SQL(context_id, 'SELECT * FROM test_desc_cols', 2);
-- Set the return value type of a column.
DBE_SQL.SET_RESULT_TYPE(context_id, 1, v_id);
DBE_SQL.SET_RESULT_TYPE(context_id, 2, v_name);
execute_ret := DBE_SQL.SQL_RUN(context_id);
loop
exit when (DBE_SQL.NEXT_ROW(context_id) <= 0);
-- Obtain values.
DBE_SQL.GET_RESULT(context_id, 1, v_id);
DBE_SQL.GET_RESULT(context_id, 2, v_name);
-- Output the result.
dbe_output.print_line('id:'|| v_id || ' name:' || v_name);
```



```
end loop;
DBE_SQL.SQL_UNREGISTER_CONTEXT(context_id);
END;
/
-- Expected result:
id:1 name:xiaoming
id:2 name:xiaohong
id:3 name:xiaolan
ANONYMOUS BLOCK EXECUTE

-- Clean the environment.
drop table if exists test_desc_cols;
DROP TABLE

-- Example 2
CREATE OR REPLACE PROCEDURE test_square(n NUMBER, square OUT NUMBER) IS
BEGIN
  square := n * n;
END;
/
CREATE PROCEDURE

DECLARE
cur NUMBER;
query varchar(2000);
ret integer;
n NUMBER;
square Integer;
BEGIN
  n := 2;
  cur := DBE_SQL.REGISTER_CONTEXT();
  query := 'BEGIN test_square(:n_bnd, :square_bnd); END;';
  DBE_SQL.SQL_SET_SQL(cur, query, 2);
  DBE_SQL.SQL_BIND_VARIABLE(cur, 'n_bnd', n);
  DBE_SQL.SQL_BIND_VARIABLE(cur, 'square_bnd', square);
  ret := DBE_SQL.SQL_RUN(cur);
  DBE_SQL.GET_VARIABLE_RESULT(cur, 'square_bnd', square);
  DBE_OUTPUT.PRINT_LINE('square = ' || square);
  DBE_SQL.SQL_UNREGISTER_CONTEXT(cur);
END;
/
-- Expected result:
square = 4
ANONYMOUS BLOCK EXECUTE

-- Clean the environment.
drop PROCEDURE test_square;
DROP PROCEDURE

-- Example 3
-- Examples of executing DESCRIBE_COLUMNS, RUN_AND_NEXT, and LAST_ROW_COUNT
-- Create a table and insert data into the table.
CREATE TABLE test_desc_cols(
    id NUMBER,
    name VARCHAR2(50)
);
INSERT INTO test_desc_cols(id, name) VALUES (1, 'xiaoming');
INSERT INTO test_desc_cols(id, name) VALUES (2, 'xiaohong');
INSERT INTO test_desc_cols(id, name) VALUES (3, 'xiaolan');

-- Create a stored procedure for printing column description information.
CREATE OR REPLACE PROCEDURE print_rec(
    rec in DBE_SQL.DESC_REC
)package AS
BEGIN
    raise INFO 'col_type            =    %', rec.col_type;
    raise INFO 'col_name            =    %', rec.col_name;
    raise INFO 'col_name_len        =    %', rec.col_name_len;
END;
```



```
/
CREATE PROCEDURE

-- Verify functions.
DECLARE
context_id  INTEGER;
col_cnt     INTEGER;
rec_tab     DBE_SQL.DESC_TAB;
excute_ret  INTEGER;
nextrow_ret INTEGER;
last_row_count INTEGER;
BEGIN
-- Open a cursor.
context_id := DBE_SQL.REGISTER_CONTEXT();
-- Compile the cursor.
DBE_SQL.SQL_SET_SQL(context_id, 'SELECT * FROM test_desc_cols', 2);
-- Print the column description information.
DBE_SQL.DESCRIBE_COLUMNS(context_id, col_cnt, rec_tab);
FOR var IN 1..col_cnt LOOP
    print_rec(rec_tab(var));
END LOOP;
-- Execute and obtain a row of data.
excute_ret := DBE_SQL.RUN_AND_NEXT(context_id);
-- Obtain a row of data.
nextrow_ret := DBE_SQL.NEXT_ROW(context_id);
-- Obtain the number of obtained data rows.
last_row_count := DBE_SQL.LAST_ROW_COUNT;
DBE_OUTPUT.PRINT_LINE('last_row_count =  '|| last_row_count);
DBE_SQL.SQL_UNREGISTER_CONTEXT(context_id);
END;
/
-- Expected result:
INFO:  col_type           =    1700
CONTEXT:  SQL statement "CALL print_rec(rec_tab[var])"
PL/pgSQL function inline_code_block line 16 at PERFORM
INFO:  col_name           =    id
CONTEXT:  SQL statement "CALL print_rec(rec_tab[var])"
PL/pgSQL function inline_code_block line 16 at PERFORM
INFO:  col_name_len       =    2
CONTEXT:  SQL statement "CALL print_rec(rec_tab[var])"
PL/pgSQL function inline_code_block line 16 at PERFORM
INFO:  col_type           =    1043
CONTEXT:  SQL statement "CALL print_rec(rec_tab[var])"
PL/pgSQL function inline_code_block line 16 at PERFORM
INFO:  col_name           =    name
CONTEXT:  SQL statement "CALL print_rec(rec_tab[var])"
PL/pgSQL function inline_code_block line 16 at PERFORM
INFO:  col_name_len       =    4
CONTEXT:  SQL statement "CALL print_rec(rec_tab[var])"
PL/pgSQL function inline_code_block line 16 at PERFORM
last_row_count =  2
ANONYMOUS BLOCK EXECUTE

-- Clean the environment.
drop table test_desc_cols;
DROP TABLE

-- Example 4
DROP TABLE if exists dbe_sql_tab;
create table dbe_sql_tab(a char(30), b int, c text, d raw, e bytea, f text, g bool);
insert into dbe_sql_tab values('aaa', 10, 'abcdefghijklmn', HEXTORAW('DEADBEEF'), 'a', 'abcdefghijklmn',
true);

DECLARE
cursorid int;
execute_ret int;
query varchar(2000);
err numeric;
v_char char(30);
```



```
v_int int;
v_long text;
v_long_len int;
v_raw raw;
v_raw_len int;
v_bytea bytea;
v_text text;
BEGIN
query := 'select * from dbe_sql_tab';
cursorid := DBE_SQL.register_context();
DBE_SQL.sql_set_sql(cursorid, query, 1);
DBE_SQL.SET_RESULT_TYPE_CHAR(cursorid, 1, v_char, 30);
DBE_SQL.SET_RESULT_TYPE_INT(cursorid, 2);
DBE_SQL.SET_RESULT_TYPE_LONG(cursorid, 3);
DBE_SQL.SET_RESULT_TYPE_RAW(cursorid, 4, v_raw, 68);
DBE_SQL.SET_RESULT_TYPE_BYTEA(cursorid, 5, v_bytea, 68);
DBE_SQL.SET_RESULT_TYPE_TEXT(cursorid, 6, 1024);
execute_ret := DBE_SQL.sql_run(cursorid);
loop
exit when (DBE_SQL.next_row(cursorid) <= 0);
DBE_SQL.GET_RESULT_CHAR(cursorid, 1, v_char);
v_int := DBE_SQL.GET_RESULT_INT(cursorid, 2);
DBE_SQL.GET_RESULT_LONG(cursorid, 3, 3, 3, v_long, v_long_len);
DBE_SQL.GET_RESULT_RAW(cursorid, 4, v_raw, err, v_raw_len);
v_bytea := DBE_SQL.GET_RESULT_BYTEA(cursorid, 5);
v_text := DBE_SQL.GET_RESULT_TEXT(cursorid, 6);
dbe_output.print_line('a:'|| v_char);
dbe_output.print_line('b:'|| v_int);
dbe_output.print_line('c:'|| v_long);
dbe_output.print_line('d:'|| v_raw);
raise info 'e:%', v_bytea;
dbe_output.print_line('f:'|| v_text);
end loop;
DBE_SQL.sql_unregister_context(cursorid);
END;
/
-- Expected result:
a:aaa
b:10
c:cde
d:DEADBEEF
INFO:  e:\x61
f:abcdefghijklmn
ANONYMOUS BLOCK EXECUTE

DECLARE
cursorid int;
execute_ret int;
query varchar(2000);
BEGIN
query := 'select * from dbe_sql_tab';
cursorid := DBE_SQL.register_context();
DBE_SQL.sql_set_sql(cursorid, query, 1);
DBE_SQL.SET_RESULT_TYPE_UNKNOWN(cursorid, 7, 'boolean');
execute_ret := DBE_SQL.sql_run(cursorid);
loop
exit when (DBE_SQL.next_row(cursorid) <= 0);
DBE_SQL.GET_RESULT_UNKNOWN(cursorid, 7, 'boolean');
end loop;
DBE_SQL.sql_unregister_context(cursorid);
END;
/
-- Expected result:
ERROR:  UnSupport data type for set_result_type(context: 8, pos: 7, 'boolean')
CONTEXT:  SQL statement "CALL pg_catalog.report_application_error('UnSupport data type for
set_result_type(context: '||context_id||', pos: '||pos||', '||PG_CATALOG.QUOTE_LITERAL(col_type)||')')"
PL/pgSQL function dbe_sql.set_result_type_unknown(integer,integer,text) line 8 at PERFORM
SQL statement "CALL dbe_sql.set_result_type_unknown(cursorid,7,'boolean')"
PL/pgSQL function inline_code_block line 10 at PERFORM
```



```
-- Clean the environment.
drop table dbe_sql_tab;
DROP TABLE

-- Example 5
drop table if exists dbe_sql_tab;
create table dbe_sql_tab(a char(30), b raw);
insert into dbe_sql_tab values('aaa', HEXTORAW('DEADBEEF'));

DECLARE
cursorid int;
execute_ret int;
query varchar(2000);
v_char char(30);
v_raw raw;
BEGIN
query := 'select * from dbe_sql_tab';
cursorid := DBE_SQL.register_context();
DBE_SQL.sql_set_sql(cursorid, query, 2);
DBE_SQL.SET_RESULT_TYPE(cursorid, 1, v_char);
DBE_SQL.SET_RESULT_TYPE_RAW(cursorid, 2, v_raw, 68);
execute_ret := DBE_SQL.sql_run(cursorid);
loop
exit when (DBE_SQL.next_row(cursorid) <= 0);
v_char := DBE_SQL.DBE_SQL_GET_RESULT_CHAR(cursorid, 1);
v_raw := DBE_SQL.DBE_SQL_GET_RESULT_RAW(cursorid, 2);
dbe_output.print_line('a:'|| v_char);
dbe_output.print_line('b:'|| v_raw);
end loop;
DBE_SQL.sql_unregister_context(cursorid);
END;
/
-- Expected result:
a:aaa
b:DEADBEEF
ANONYMOUS BLOCK EXECUTE

-- Clean the environment.
drop table dbe_sql_tab;
DROP TABLE

-- Example 6
DECLARE
cursorid int;
execute_ret int;
is_open boolean;
BEGIN
cursorid := DBE_SQL.register_context();
is_open := DBE_SQL.IS_ACTIVE(cursorid);
dbe_output.print_line('is_open:' ||is_open);
DBE_SQL.sql_unregister_context(cursorid);
is_open := DBE_SQL.IS_ACTIVE(cursorid);
dbe_output.print_line('is_open:' ||is_open);
END;
/
-- Expected result:
is_open:true
is_open:false
ANONYMOUS BLOCK EXECUTE

-- Example 7
create table tbl(a integer ,b varchar(100));
CREATE TABLE

DECLARE
c integer;
v1 integer[];
v2 varchar2[];
```



```
query varchar(2000);
ret integer;
begin
c := dbe_sql.register_context();
query := 'insert into tbl(a,b) values(:v_1, :v_2);';
dbe_sql.sql_set_sql(c, query, 2);
v1(1) := 1;
v1(2) := 2;
v2(1) := '1';
v2(2) := '2';
dbe_sql.sql_bind_array(c, 'v_1', v1);
dbe_sql.sql_bind_array(c, 'v_2', v2);
ret := dbe_sql.sql_run(c);
dbe_sql.sql_unregister_context(c);
end;
/
ANONYMOUS BLOCK EXECUTE

select * from tbl order by a;
-- Expected result:
 a | b
---+---
 1 | 1
 2 | 2
(2 rows)

-- Clean the environment.
drop table tbl;
DROP TABLE

-- Example 8
-- Prerequisites
drop table if exists dbe_sql_tab;
create table dbe_sql_tab(a int, b text, c raw, d text, e char, f int);
insert into dbe_sql_tab values(1, '9', '5', '13', 'a', 1);
insert into dbe_sql_tab values(2, '9', '6', '14', 'b', 2);
insert into dbe_sql_tab values(3, '7', '7', '15', 'c', 3);
insert into dbe_sql_tab values(4, '6', '8', '16', 'd', 4);

DECLARE
query varchar(2000);
context_id int;
execute_ret int;
v_id int;
v_ints int[];
v_texts text[];
v_raws raw[];
v_byteas bytea[];
v_chars character[];
v_type int[];
BEGIN
query := ' select * from dbe_sql_tab';
context_id := dbe_sql.register_context();
dbe_sql.sql_set_sql(context_id, query, 1);
DBE_SQL.SET_RESULT_TYPE_INTS(context_id, 1, v_ints, 3, 1);
DBE_SQL.SET_RESULT_TYPE_TEXTS(context_id, 2, v_texts, 3, 1, 1024);
DBE_SQL.SET_RESULT_TYPE_RAWS(context_id, 3, v_raws, 3, 1, 1024);
DBE_SQL.SET_RESULT_TYPE_BYTEAS(context_id, 4, v_byteas, 3, 1, 1024);
DBE_SQL.SET_RESULT_TYPE_CHARS(context_id, 5, v_chars, 3, 1, 1024);
DBE_SQL.SET_RESULTS_TYPE(context_id, 6, v_type, 3, 1);
execute_ret := dbe_sql.sql_run(context_id);
loop
v_id := dbe_sql.next_row(context_id);
v_ints := DBE_SQL.GET_RESULTS_INT(context_id, 1, v_ints);
v_texts := DBE_SQL.GET_RESULTS_TEXT(context_id, 2, v_texts);
v_raws := DBE_SQL.GET_RESULTS_RAW(context_id, 3, v_raws);
v_byteas := DBE_SQL.GET_RESULTS_BYTEA(context_id, 4, v_byteas);
v_chars := DBE_SQL.GET_RESULTS_CHAR(context_id, 5, v_chars);
v_type := DBE_SQL.GET_RESULTS(context_id, 6, v_type);
```



```
exit when(v_id != 3);
end loop;
FOR i IN v_ints.FIRST .. v_ints.LAST  LOOP
        dbe_output.print_line('int' || i || ' = ' || v_ints[i]);
END LOOP;
FOR i IN v_texts.FIRST .. v_texts.LAST  LOOP
        dbe_output.print_line('text' || i || ' = ' || v_texts[i]);
END LOOP;
FOR i IN v_raws.FIRST .. v_raws.LAST  LOOP
        dbe_output.print_line('raw' || i || ' = ' || v_raws[i]);
END LOOP;
FOR i IN v_byteas.FIRST .. v_byteas.LAST  LOOP
        dbe_output.print_line('bytea' || i || ' = ' || v_byteas[i]);
END LOOP;
FOR i IN v_chars.FIRST .. v_chars.LAST  LOOP
        dbe_output.print_line('char' || i || ' = ' || v_chars[i]);
END LOOP;
FOR i IN v_type.FIRST .. v_type.LAST  LOOP
        dbe_output.print_line('type' || i || ' = ' || v_type[i]);
END LOOP;
dbe_sql.sql_unregister_context(context_id);
END;
/
-- Expected result:
int1 = 1
int2 = 2
int3 = 3
int4 = 4
text1 = 9
text2 = 9
text3 = 7
text4 = 6
raw1 = 05
raw2 = 06
raw3 = 07
raw4 = 08
bytea1 = \x3133
bytea2 = \x3134
bytea3 = \x3135
bytea4 = \x3136
char1 = a
char2 = b
char3 = c
char4 = d
type1 = 1
type2 = 2
type3 = 3
type4 = 4
ANONYMOUS BLOCK EXECUTE

-- Clean the environment.
drop table if exists dbe_sql_tab;
DROP TABLE

-- Example 9
-- Prerequisites
drop table if exists dbe_sql_tab;
create table dbe_sql_tab(a int ,b int);
insert into dbe_sql_tab values(1,3);

DECLARE
context_id int;
type re_rssc is record (col_num int, desc_col dbe_sql.desc_tab);
employer re_rssc;
res re_rssc;
d int;
dd dbe_sql.desc_tab;
query varchar(2000);
BEGIN
```



```
query := 'select * from dbe_sql_tab';
-- Open a cursor.
context_id := dbe_sql.register_context();
-- Compile the cursor.
dbe_sql.sql_set_sql(context_id, query, 1);
-- Execute the cursor.
res := dbe_sql.sql_describe_columns(context_id, d,dd);
-- Output the result.
dbe_output.print_line('col_num:' || res.col_num);
dbe_output.print_line('col_type:' || res.desc_col[1].col_type);
dbe_output.print_line('col_max_len:' || res.desc_col[1].col_max_len);
dbe_output.print_line('col_name:' || res.desc_col[1].col_name);
dbe_output.print_line('col_name_len:' || res.desc_col[1].col_name_len);
dbe_output.print_line('col_schema_name:' || res.desc_col[1].col_schema_name);
dbe_output.print_line('col_schema_name_len:' || res.desc_col[1].col_schema_name_len);
dbe_output.print_line('col_precision:' || res.desc_col[1].col_precision);
dbe_output.print_line('col_scale:' || res.desc_col[1].col_scale);
dbe_output.print_line('col_charsetid:' || res.desc_col[1].col_charsetid);
dbe_output.print_line('col_charsetform:' || res.desc_col[1].col_charsetform);
dbe_output.print_line('col_null_ok:' || res.desc_col[1].col_null_ok);
-- Close the cursor.
dbe_sql.sql_unregister_context(context_id);
END;
/
-- Expected result:
col_num:2
col_type:23
col_max_len:4
col_name:a
col_name_len:1
col_schema_name:
col_schema_name_len:0
col_precision:0
col_scale:0
col_charsetid:0
col_charsetform:0
col_null_ok:true
ANONYMOUS BLOCK EXECUTE

-- Clean the environment.
drop table if exists dbe_sql_tab;
DROP TABLE

-- Example 10
drop table if exists dbe_sql_tab;
create table dbe_sql_tab(a int);
insert into dbe_sql_tab values(1);
insert into dbe_sql_tab values(2);
insert into dbe_sql_tab values(3);

DECLARE
query varchar(2000);
context_id int;
execute_ret int;
v_id int;
v_ints int[];
i1 integer;
BEGIN
query := 'select * from dbe_sql_tab';
context_id := dbe_sql.register_context();
dbe_sql.sql_set_sql(context_id, query, 1);
DBE_SQL.SQL_SET_RESULTS_TYPE_C(context_id, 1, v_ints, 3, 1, i1, 0);
execute_ret := dbe_sql.sql_run(context_id);
loop
v_id := dbe_sql.next_row(context_id);
v_ints := DBE_SQL.SQL_GET_VALUES_C(context_id, 1, v_ints, i1);
exit when(v_id != 3);
end loop;
FOR i IN v_ints.FIRST .. v_ints.LAST  LOOP
```



```
        dbe_output.print_line('int' || i || ' = ' || v_ints[i]);
END LOOP;
dbe_sql.sql_unregister_context(context_id);
END;
/
-- Expected result:
int1 = 1
int2 = 2
int3 = 3
ANONYMOUS BLOCK EXECUTE

-- Clean the environment.
drop table if exists dbe_sql_tab;
DROP TABLE

-- Example 11
CREATE OR REPLACE PROCEDURE test_proc(out_int out Integer, out_char out char, out_raw out raw,
out_text out text) IS
BEGIN
  out_int := 1;
  out_char := 'a';
  out_raw := 'b';
  out_text := 'c';
END;
/
CREATE PROCEDURE

DECLARE
cur NUMBER;
query varchar(2000);
ret integer;
v_int Integer;
v_char char;
v_raw raw;
v_text text;
BEGIN
  cur := DBE_SQL.REGISTER_CONTEXT();
  query := 'BEGIN test_proc(:v_int, :v_char, :v_raw, :v_text); END;';
  DBE_SQL.SQL_SET_SQL(cur, query, 2);
  DBE_SQL.SQL_BIND_VARIABLE(cur, 'v_int', v_int);
  DBE_SQL.SQL_BIND_VARIABLE(cur, 'v_char', v_char, 1024);
  DBE_SQL.SQL_BIND_VARIABLE(cur, 'v_raw', v_raw, 1024);
  DBE_SQL.SQL_BIND_VARIABLE(cur, 'v_text', v_text, 1024);
  ret := DBE_SQL.SQL_RUN(cur);
  DBE_SQL.GET_VARIABLE_RESULT_INT(cur, 'v_int', v_int);
  v_char := DBE_SQL.GET_VARIABLE_RESULT_CHAR(cur, 'v_char');
  DBE_SQL.GET_VARIABLE_RESULT_RAW(cur, 'v_raw', v_raw);
  v_text := DBE_SQL.GET_VARIABLE_RESULT_TEXT(cur, 'v_text');
  DBE_OUTPUT.PRINT_LINE('v_int = ' || v_int);
  DBE_OUTPUT.PRINT_LINE('v_char = ' || v_char);
  DBE_OUTPUT.PRINT_LINE('v_raw = ' || v_raw);
  DBE_OUTPUT.PRINT_LINE('v_text = ' || v_text);
  DBE_SQL.SQL_UNREGISTER_CONTEXT(cur);
END;
/
-- Expected result:
v_int = 1
v_char = a
v_raw = 0B
v_text = c
ANONYMOUS BLOCK EXECUTE

-- Clean the environment.
drop procedure test_proc;
DROP PROCEDURE

-- Example 12
CREATE OR REPLACE PROCEDURE test_proc(out_int out Integer[], out_char out char[], out_raw out raw[],
out_text out text[]) IS
```



```
BEGIN
  out_int(0) := 1;
  out_char(0) := 'a';
  out_raw(0) := 'b';
  out_text(0) := 'c';
END;
/
CREATE PROCEDURE

DECLARE
cur NUMBER;
query varchar(2000);
ret integer;
v_int Integer[];
v_char char[];
v_raw raw[];
v_text text[];
BEGIN
  cur := DBE_SQL.REGISTER_CONTEXT();
  query := 'call test_proc(:v_int, :v_char, :v_raw, :v_text);';
  DBE_SQL.SQL_SET_SQL(cur, query, 1);
  DBE_SQL.SQL_BIND_ARRAY(cur, 'v_int', v_int);
  DBE_SQL.SQL_BIND_ARRAY(cur, 'v_char', v_char);
  DBE_SQL.SQL_BIND_ARRAY(cur, 'v_raw', v_raw);
  DBE_SQL.SQL_BIND_ARRAY(cur, 'v_text', v_text);
  ret := DBE_SQL.SQL_RUN(cur);
  DBE_SQL.GET_ARRAY_RESULT_INT(cur, 'v_int', v_int);
  DBE_SQL.GET_ARRAY_RESULT_CHAR(cur, 'v_char', v_char);
  DBE_SQL.GET_ARRAY_RESULT_RAW(cur, 'v_raw', v_raw);
  DBE_SQL.GET_ARRAY_RESULT_TEXT(cur, 'v_text', v_text);
  DBE_OUTPUT.PRINT_LINE('v_int = ' || v_int(0));
  DBE_OUTPUT.PRINT_LINE('v_char = ' || v_char(0));
  DBE_OUTPUT.PRINT_LINE('v_raw = ' || v_raw(0));
  DBE_OUTPUT.PRINT_LINE('v_text = ' || v_text(0));
  DBE_SQL.SQL_UNREGISTER_CONTEXT(cur);
END;
/
-- Expected result:
v_int = 1
v_char = a
v_raw = 0B
v_text = c
ANONYMOUS BLOCK EXECUTE

-- Clean the environment.
drop PROCEDURE test_proc;
DROP PROCEDURE
```

10.12.2.16 DBE_STATS

The advanced package DBE_STATS implements some capabilities of managing statistics information, including locking, unlocking, rolling back, and clearing historical statistics.

NOTE

The **DBE_STATS** advanced package does not support temporary tables.

API Description

Table 10-368 DBE_STATS overview

API	Description
DBE_STATS.LOCK_TABLE_STATS	Locks table-level statistics.
DBE_STATS.LOCK_PARTITION_STATS	Locks partition-level statistics.
DBE_STATS.LOCK_COLUMN_STATS	Locks column-level statistics.
DBE_STATS.LOCK_SCHEMA_STATS	Locks statistics about all tables in a specified schema.
DBE_STATS.UNLOCK_TABLE_STATS	Unlocks table-level statistics.
DBE_STATS.UNLOCK_PARTITION_STATS	Unlocks partition-level statistics.
DBE_STATS.UNLOCK_COLUMN_STATS	Unlocks column-level statistics.
DBE_STATS.UNLOCK_SCHEMA_STATS	Unlocks statistics about all tables in a specified schema.
DBE_STATS.RESTORE_TABLE_STATS	Rolls back table-level statistics to a specified time point.
DBE_STATS.RESTORE_PARTITION_STATS	Rolls back partition-level statistics to a specified time point.
DBE_STATS.RESTORE_COLUMN_STATS	Rolls back column-level statistics to a specified time point.
DBE_STATS.RESTORE_SCHEMA_STATS	Rolls back statistics about all tables in a specified schema to a specified time point.
DBE_STATS.PURGE_STATS	Clears all historical statistics before a specified time point.
DBE_STATS.GET_STATS_HISTORY_RETENTION	Obtains the retention period of historical statistics.
DBE_STATS.GET_STATS_HISTORY_AVAILABILITY	Obtains the earliest available time of the current historical statistics.
DBE_STATS.CREATE_STAT_TABLE	Creates a statistics table for storing statistics.
DBE_STATS.DROP_STAT_TABLE	Deletes a statistics table for storing statistics.

API	Description
DBE_STATS.EXPORT_INDEX_STATS	Retrieves statistics for the specified index and stores them in the user statistics table.
DBE_STATS.EXPORT_TABLE_STATS	Retrieves statistics for the specified table and stores them in the user statistics table.
DBE_STATS.EXPORT_COLUMN_STATS	Retrieves statistics for the specified column and stores them in the user statistics table.
DBE_STATS.EXPORT_SCHEMA_STATS	Retrieves statistics for the specified schema and stores them in the user statistics table.
DBE_STATS.IMPORT_INDEX_STATS	Retrieves statistics for the specified index from the user statistics table and writes it back to the system catalog.
DBE_STATS.IMPORT_TABLE_STATS	Retrieves statistics for the specified table from the user statistics table and writes it back to the system catalog.
DBE_STATS.IMPORT_COLUMN_STATS	Retrieves statistics for the specified column from the user statistics table and writes it back to the system catalog.
DBE_STATS.IMPORT_SCHEMA_STATS	Retrieves statistics for the specified schema from the user statistics table and writes it back to the system catalog.
DBE_STATS.SET_COLUMN_STATS	Sets column-related statistics, including single-column, multi-column, and expression statistics.
DBE_STATS.SET_INDEX_STATS	Sets index-related statistics.
DBE_STATS.SET_TABLE_STATS	Sets table-related statistics.
DBE_STATS.DELETE_COLUMN_STATS	Deletes column-related statistics, including single-column, multi-column, and expression statistics.
DBE_STATS.DELETE_INDEX_STATS	Deletes index-related statistics.
DBE_STATS.DELETE_TABLE_STATS	Deletes table-related statistics.
DBE_STATS.DELETE_SCHEMA_STATS	Deletes statistics about all tables, indexes, and columns in a schema.

- DBE_STATS.LOCK_TABLE_STATS

  Locks the statistics of a table. After the table is locked, the statistics of the table cannot be updated, and the indexes, partitions, and columns involved in the table are locked synchronously. This API does not return any value.

The prototype of the DBE_STATS.LOCK_TABLE_STATS function is as follows:

```
DBE_STATS.LOCK_TABLE_STATS(
   ownname   VARCHAR2,
   tabname   VARCHAR2
);
```

Table 10-369 DBE_STATS.LOCK_TABLE_STATS API

Parameter	Type	Whether NULL Is Allowed	Description
ownname	varchar2	Yes	Name of the schema to which the table to be locked belongs. Null indicates that the current schema is used by default.
tabname	varchar2	No	Name of the table to be locked.

NOTE

- When using this API, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- You can check stat_state in reloptions of the table to obtain the lock status of the table.
- When a table is locked, its statistics cannot be updated.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- Create a table.
gaussdb=# CREATE SCHEMA dbe_stats_lock;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA=dbe_stats_lock;
SET
gaussdb=# CREATE TABLE t1(a int,b int);
CREATE TABLE

-- Lock the table and check its locking status.
gaussdb=# CALL DBE_STATS.LOCK_TABLE_STATS(ownname=>'dbe_stats_lock',tabname=>'t1');
 lock_table_stats
------------------

(1 row)
gaussdb=# SELECT relname,instr(reloptions::text,'stat_state=locked',1,1) <> 0  as exist_lock FROM
PG_CLASS WHERE relname='t1' and relnamespace = (SELECT oid FROM PG_NAMESPACE WHERE
nspname='dbe_stats_lock');
 relname | exist_lock
---------+------------
 t1      | t
(1 row)

-- An error is reported after the ANALYZE operation is performed.
gaussdb=# ANALYZE t1;
ERROR:  The statistics is locked, cannot be updated.
```



```
-- Delete a table or namespace.
gaussdb=# DROP TABLE t1;
DROP TABLE
gaussdb=# DROP SCHEMA dbe_stats_lock;
DROP SCHEMA
```

- DBE_STATS.LOCK_PARTITION_STATS

  Locks the statistics of a partition. After the partition is locked, the statistics of the partition cannot be updated, and the indexes, partitions, and columns related to the partition are locked synchronously. This API does not return any value.

  The prototype of the DBE_STATS.LOCK_PARTITION_STATS function is as follows:

```
DBE_STATS.LOCK_PARTITION_STATS(
    ownname     VARCHAR2,
    tabname     VARCHAR2,
    partname    VARCHAR2
);
```

Table 10-370 DBE_STATS.LOCK_PARTITION_STATS API

Parameter	Type	Whether NULL Is Allowed	Description
ownname	varchar2	Yes	Name of the schema to which the partition to be locked belongs. Null indicates that the current schema is used by default.
tabname	varchar2	No	Name of the table to which the partition to be locked belongs.
partname	varchar2	No	Name of the partition to be locked.

NOTE

- When using this API, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- You can view stat_state in reloptions of the partition to obtain the lock status of the table.
- When the partition or the table to which the table belongs is locked, its statistics cannot be updated.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- Create a table.
gaussdb=# CREATE SCHEMA dbe_stats_lock;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA=dbe_stats_lock;
SET
gaussdb=# CREATE TABLE upart_table(a int, b int, c int) PARTITION BY RANGE(a)
```



```
 (
    PARTITION p1 VALUES LESS THAN(1200),
    PARTITION p2 VALUES LESS THAN(2400),
    PARTITION p3 VALUES LESS THAN(MAXVALUE)
 );
CREATE TABLE

-- Lock a partition. Other partitions and tables are not affected.
gaussdb=# CALL
DBE_STATS.LOCK_PARTITION_STATS(ownname=>'dbe_stats_lock',tabname=>'upart_table',partname=>'
p1');
 lock_partition_stats
----------------------

(1 row)

gaussdb=# SELECT relname,instr(reloptions::text,'stat_state=locked',1,1) <> 0 as exist_lock FROM
PG_CLASS WHERE relname='upart_table';
   relname   | exist_lock
-------------+------------
 upart_table | f
(1 row)

gaussdb=# SELECT relname,instr(reloptions::text,'stat_state=locked',1,1) <> 0 as exist_lock FROM
PG_PARTITION WHERE parentid='upart_table'::REGCLASS;
   relname   | exist_lock
-------------+------------
 upart_table | f
 p2          | f
 p3          | f
 p1          | t
(4 rows)

-- Delete a table or namespace.
gaussdb=# DROP TABLE upart_table;
DROP TABLE
gaussdb=# DROP SCHEMA dbe_stats_lock;
DROP SCHEMA
```

- DBE_STATS.LOCK_COLUMN_STATS

  Locks the statistics of a column. After the column is locked, the statistics of the column cannot be updated. This API does not return any value.

  The prototype of the DBE_STATS.LOCK_COLUMN_STATS function is as follows:

```
DBE_STATS.LOCK_COLUMN_STATS(
   ownname     VARCHAR2,
   tabname     VARCHAR2,
   colname     VARCHAR2,
   partname    VARCHAR2 DEFAULT NULL
);
```

Table 10-371 DBE_STATS.LOCK_COLUMN_STATS API

Parameter	Type	Whether NULL Is Allowed	Description
ownname	varchar2	Yes	Name of the schema to which the column to be locked belongs. Null indicates that the current schema is used by default.

Parameter	Type	Whether NULL Is Allowed	Description
tabname	varchar2	No	Name of the table to which the column to be locked belongs.
colname	varchar2	No	Name of the column to be locked.
partname	varchar2	Yes	Name of the partition to which the column to be locked belongs. The default value is **NULL**.

NOTE

- When using this API, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- If **partname** is **NULL**, the list-level statistics are locked by default.
- If the column-level statistics of the table do not exist, the locking can be executed successfully but does not take effect.
- You can view **stastate** in **PG_STATISTIC**/**PG_STATISTIC_EXT** to obtain the lock status.
- Before locking multiple columns, query the **staextname** column in **PG_STATISTIC_EXT** to obtain the aliases of multiple columns and transfer them as the input parameter **colname**.
- If a column is locked or the table or partition to which the column belongs is locked, the statistics cannot be updated. The locked information about the column is not displayed. You need to view the locked information by using logging_module of DBE_STATS at the DEBUG2 level in logs.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- Create a table and collect statistics.
gaussdb=# CREATE SCHEMA dbe_stats_lock;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA=dbe_stats_lock;
SET
gaussdb=# CREATE TABLE t1(a int,b int);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(generate_series(1,100),1);
INSERT 0 100
gaussdb=# ANALYZE t1;
ANALYZE

-- After locking a column, check the locking status of the column.
gaussdb=# CALL
DBE_STATS.LOCK_COLUMN_STATS(ownname=>'dbe_stats_lock',tabname=>'t1',colname=>'a');
 lock_column_stats
-------------------

(1 row)

gaussdb=# SELECT staattnum,stastate FROM PG_STATISTIC WHERE starelid='t1'::REGCLASS;
 staattnum | stastate
```



```
-----------+----------
         2 | u
         1 | l
(2 rows)

-- Delete a table or namespace.
gaussdb=# DROP TABLE t1;
DROP TABLE
gaussdb=# DROP SCHEMA dbe_stats_lock;
DROP SCHEMA
```

- DBE_STATS.LOCK_SCHEMA_STATS

  Locks statistics about all tables in a specified schema. This API does not return any value.

  The prototype of the DBE_STATS.LOCK_SCHEMA_STATS function is as follows:

```
DBE_STATS.LOCK_SCHEMA_STATS(
    ownname     VARCHAR2
);
```

Table 10-372 DBE_STATS.LOCK_SCHEMA_STATS API

Parameter	Type	Whether NULL Is Allowed	Description
ownname	varchar2	Yes	Name of the specified schema. Null indicates that the current schema is used by default.

NOTE

- When this API is called, all tables on which the current user has the ANALYZE permission in the schema are locked. Tables on which the current user does not have the ANALYZE permission are skipped.
- To call this API, you must have the USAGE permission on the specified schema.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- The method of locking schema statistics is the same as that of locking table statistics. The
following describes only the API calling method. For details, see the example of
DBE_STATS.LOCK_TABLE_STA....
gaussdb=# CALL DBE_STATS.LOCK_SCHEMA_STATS(ownname=>'dbe_stats_lock');
 lock_schema_stats
-------------------


(1 row)
```

- DBE_STATS.UNLOCK_TABLE_STATS

  Unlocks the statistics of a table. After the table is unlocked, the statistics of the table can be updated. The indexes, partitions, and columns involved in the table are unlocked synchronously. This API does not return any value.

  The prototype of the DBE_STATS.UNLOCK_TABLE_STATS function is as follows:

```
DBE_STATS.UNLOCK_TABLE_STATS(
   ownname    VARCHAR2,
```

```
  tabname   VARCHAR2
);
```

Table 10-373 DBE_STATS.UNLOCK_TABLE_STATS API

Parameter	Type	Whether NULL Is Allowed	Description
ownname	varchar2	Yes	Name of the schema to which the table to be unlocked belongs. Null indicates that the current schema is used by default.
tabname	varchar2	No	Name of the table to be unlocked.

NOTE

- When using this API, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- You can check stat_state in reloptions of the table to obtain the lock status of the table.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- The method of unlocking table statistics is the same as that of locking table statistics. The following describes only the API calling method. For details, see the DBE_STATS.LOCK_TABLE_STA... example.
gaussdb=# CALL DBE_STATS.UNLOCK_TABLE_STATS(ownname=>'dbe_stats_lock',tabname=>'t1');
 unlock_table_stats
--------------------


(1 row)
```

- DBE_STATS.UNLOCK_PARTITION_STATS

  Unlocks the statistics of a partition. After the partition is unlocked, the statistics of the partition can be updated. The indexes, partitions, and columns involved in the partition are unlocked synchronously. This API does not return any value.

  The prototype of the DBE_STATS.UNLOCK_PARTITION_STATS function is as follows:

```
DBE_STATS.UNLOCK_PARTITION_STATS(
   ownname     VARCHAR2,
   tabname     VARCHAR2,
   partname    VARCHAR2
);
```


Table 10-374 DBE_STATS.UNLOCK_PARTITION_STATS API

Parameter	Type	Whether NULL Is Allowed	Description
ownname	varchar2	Yes	Name of the schema to which the partition to be unlocked belongs. Null indicates that the current schema is used by default.
tabname	varchar2	No	Name of the table to which the partition to be unlocked belongs.
partname	varchar2	No	Name of the partition to be unlocked.

NOTE

- When using this API, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- You can view stat_state in reloptions of the partition to obtain the lock status of the table.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- The method of unlocking partition statistics is the same as that of locking partition statistics. The following describes only the API calling method. For details, see the DBE_STATS.LOCK_PARTIION_... example.
gaussdb=# CALL DBE_STATS.UNLOCK_PARTITION_STATS(ownname=>'dbe_stats_lock',tabname=>'upart_table',partname=>'p1');
 unlock_partition_stats
------------------------

(1 row)
```

- DBE_STATS.UNLOCK_COLUMN_STATS

  Unlocks the statistics of a column. After being unlocked, the statistics of the column can be updated. This API does not return any value.

  The prototype of the DBE_STATS.UNLOCK_COLUMN_STATS function is as follows:

```
DBE_STATS.LOCK_COLUMN_STATS(
    ownname     VARCHAR2,
    tabname     VARCHAR2,
    colname     VARCHAR2,
    partname    VARCHAR2 DEFAULT NULL
);
```


Table 10-375 DBE_STATS.UNLOCK_COLUMN_STATS API

Parameter	Type	Whether NULL Is Allowed	Description
ownname	varchar2	Yes	Name of the schema to which the column to be unlocked belongs. Null indicates that the current schema is used by default.
tabname	varchar2	No	Name of the table to which the column to be unlocked belongs.
colname	varchar2	No	Name of the column to be unlocked.
partname	varchar2	Yes	Name of the partition to which the column to be unlocked belongs. The default value is **NULL**.

NOTE

- When using this API, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- If **partname** is **NULL**, the list-level statistics and column-level statistics of the cascaded partition are unlocked by default.
- You can view the stastate bit in **PG_STATISTIC**/**PG_STATISTIC_EXT** to obtain the lock status.
- Before unlocking multiple columns, query the **staextname** column in **PG_STATISTIC_EXT** to obtain the aliases of multiple columns and transfer them as the input parameter **colname**.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- The method of unlocking column statistics is the same as that of locking column statistics. The following describes only the API calling method. For details, see the DBE_STATS.LOCK_COLUMN_STATS example.
gaussdb=# CALL DBE_STATS.UNLOCK_COLUMN_STATS(ownname=>'dbe_stats_lock',tabname=>'t1',colname=>'a');
 unlock_column_stats
---------------------


(1 row)
```

- DBE_STATS.UNLOCK_SCHEMA_STATS

  Locks statistics about all tables in a specified schema. This API does not return any value.

  The prototype of the DBE_STATS.UNLOCK_SCHEMA_STATS function is as follows:

```
DBE_STATS.UNLOCK_SCHEMA_STATS(
   ownname    VARCHAR2
);
```


Table 10-376 DBE_STATS.UNLOCK_SCHEMA_STATS API

Parameter	Type	Whether NULL Is Allowed	Description
ownname	varchar2	Yes	Name of the specified schema. Null indicates that the current schema is used by default.

NOTE

- When this API is called, all tables on which the current user has the ANALYZE permission in the schema are unlocked. Tables on which the current user does not have the ANALYZE permission are skipped.
- To call this API, you must have the USAGE permission on the specified schema.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- The method of unlocking schema statistics is the same as that of unlocking table statistics. The following describes only the API calling method. For details, see the DBE_STATS.LOCK_TABLE_STATS example.
gaussdb=# CALL DBE_STATS.UNLOCK_SCHEMA_STATS(ownname=>'dbe_stats_lock');
 unlock_schema_stats
---------------------

(1 row)
```

- DBE_STATS.RESTORE_TABLE_STATS

  Rolls back table statistics to a specified time point. The last statistics collected before the time point are loaded to the system catalog, and the statistics of the lower-level partitions, indexes, and columns that are not locked are cascaded. This API does not return any value.

  The prototype of the DBE_STATS.RESTORE_TABLE_STATS function is as follows:

```
DBE_STATS.RESTORE_TABLE_STATS(
    ownname                 VARCHAR2,
    tabname                 VARCHAR2,
    as_of_timestamp         TIMESTAMPTZ,
    restore_cluster_index   BOOLEAN DEFAULT NULL,
    force                   BOOLEAN DEFAULT FALSE,
    no_invalidate           BOOLEAN DEFAULT NULL
);
```


Table 10-377 DBE_STATS.RESTORE_TABLE_STATS API

Parameter	Type	Whether NULL Is Allowed	Description
ownname	VARCHAR2	Yes	Name of the schema to which the table to be rolled back belongs. Null indicates that the current schema is used by default.
tabname	VARCHAR2	No	Name of the table to be rolled back.
as_of_timestamp	timestamp with time zone	No	Target time point to be rolled back.
restore_cluster_index	BOOLEAN	Yes	Not supported currently.
force	BOOLEAN	Yes	Determines whether to forcibly roll back the statistics when the statistics are locked. The default value is **FALSE**.
no_invalidate	BOOLEAN	Yes	Not supported currently.

NOTE

- When using this API, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- When this API is called, the statistics of its lower-level partitions, indexes, and columns are rolled back at the same time. If the lower-level partitions or columns are locked, the statistics are skipped.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- Create a table and collect statistics twice.
gaussdb=# CREATE SCHEMA dbe_stats_restore;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA=dbe_stats_restore;
SET
gaussdb=# CREATE TABLE t1(a int, b int);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(1,1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES(1,1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES(1,1);
INSERT 0 1
gaussdb=# ANALYZE t1;
ANALYZE
gaussdb=# INSERT INTO t1 VALUES(2,2);
```



```
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES(2,2);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES(2,2);
INSERT 0 1
gaussdb=# ANALYZE t1;
ANALYZE

-- View history tables.
gaussdb=# SELECT relname,reltuples FROM GS_TABLESTATS_HISTORY WHERE relname='t1';
 relname | reltuples
---------+-----------
 t1      |         3
 t1      |         6
(2 rows)

-- Display statistics in the current system catalog.
gaussdb=# SELECT relname,reltuples FROM PG_CLASS WHERE relname='t1' AND relnamespace =
(SELECT oid FROM PG_NAMESPACE WHERE nspname='dbe_stats_restore');
 relname | reltuples
---------+-----------
 t1      |         6
(1 row)

-- Roll back to the earliest statistics and check the system catalog.
gaussdb=# CALL
DBE_STATS.RESTORE_TABLE_STATS(ownname=>'dbe_stats_restore',tabname=>'t1',as_of_timestamp=>(
(SELECT MIN(reltimestamp) FROM GS_TABLESTATS_HISTORY WHERE relname='t1') + INTERVAL '1
second'));
 restore_table_stats
---------------------


(1 row)

gaussdb=# SELECT relname,reltuples FROM PG_CLASS WHERE relname='t1' AND relnamespace =
(SELECT oid FROM PG_NAMESPACE WHERE nspname='dbe_stats_restore');
 relname | reltuples
---------+-----------
 t1      |         3
(1 row)

-- Delete a table or namespace.
gaussdb=# DROP TABLE t1;
DROP TABLE
gaussdb=# DROP SCHEMA dbe_stats_restore;
DROP SCHEMA
```

- DBE_STATS.RESTORE_PARTITION_STATS

  Rolls back partition statistics to a specified time point. The last statistics collected before the time point are loaded to the system catalog, and the statistics of the lower-level partitions, indexes, and columns that are not locked are cascaded. This API does not return any value.

  The prototype of the DBE_STATS.RESTORE_PARTITION_STATS function is as follows:

```
DBE_STATS.RESTORE_PARTITION_STATS(
    ownname                 VARCHAR2,
    tabname                 VARCHAR2,
    partname                VARCHAR2,
    as_of_timestamp           TIMESTAMPTZ,
    force                   BOOLEAN DEFAULT FALSE,
    no_invalidate           BOOLEAN DEFAULT NULL
);
```


Table 10-378 DBE_STATS.RESTORE_PARTITION_STATS API

Parameter	Type	Whether NULL Is Allowed	Description
ownname	VARCHAR2	Yes	Name of the schema to which the partition to be rolled back belongs.
tabname	VARCHAR2	No	Name of the table to which the partition to be rolled back belongs.
partname	VARCHAR2	No	Name of the partition to be rolled back.
as_of_timestamp	timestamp with time zone	No	Target time point to be rolled back.
force	BOOLEAN	Yes	Determines whether to forcibly roll back the statistics when the statistics are locked.
no_invalidate	BOOLEAN	Yes	Not supported currently.

NOTE

- When using this API, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- When this API is called, the statistics of its lower-level partitions, indexes, and columns are rolled back.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- The method of rolling back partition statistics is the same as that of rolling back table statistics. The following describes only the API calling method. For details, see the example of DBE_STATS.RESTORE_TABLE_STATS.
gaussdb=# CALL DBE_STATS.RESTORE_PARTITION_STATS(ownname=>'dbe_stats_restore',tabname=>'t1',partname=>'p1',as_of_timestamp=>((SELECT MIN(reltimestamp) FROM GS_TABLESTATS_HISTORY WHERE relname='t1') + INTERVAL '1 second'));
 restore_partition_stats
-------------------------

(1 row)
```

- DBE_STATS.RESTORE_COLUMN_STATS

  Rolls back the column statistics to a specified time point. The last statistics collected before the time point to the system catalog are loaded. This API does not return any value.

  The prototype of the DBE_STATS.RESTORE_COLUMN_STATS function is as follows:


```
RESTORE_COLUMN_STATS(
   ownname              VARCHAR2,
   tabname              VARCHAR2,
   colname              VARCHAR2,
   partname             VARCHAR2 DEFAULT NULL,
   as_of_timestamp        TIMESTAMPTZ,
   force              BOOLEAN DEFAULT FALSE,
   no_invalidate        BOOLEAN DEFAULT NULL
);
```

Table 10-379 DBE_STATS.RESTORE_COLUMN_STATS API

Parameter	Type	Whether NULL Is Allowed	Description
ownname	VARCHAR2	Yes	Name of the schema to which the column to be rolled back belongs.
tabname	VARCHAR2	No	Name of the table to which the column to be rolled back belongs.
colname	VARCHAR2	No	Name of the column to be rolled back.
partname	VARCHAR2	Yes	Name of the partition to which the column to be rolled back belongs.
as_of_timestamp	timestamp with time zone	No	Target time point to be rolled back.
force	BOOLEAN	Yes	Determines whether to forcibly roll back the statistics when the statistics are locked.
no_invalidate	BOOLEAN	Yes	Not supported currently.

NOTE

- When using this API, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- If **partname** is **NULL**, the list-level statistics are rolled back by default.
- Before rolling back multiple columns, query the **staextname** column in **PG_STATISTIC_EXT** to obtain the aliases of multiple columns and transfer them as the input parameter **colname**.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- Create a table and collect statistics twice.
gaussdb=# CREATE SCHEMA dbe_stats_restore;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA=dbe_stats_restore;
```



```
SET
gaussdb=# CREATE TABLE t1(a int, b int);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(1,1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES(1,1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES(1,1);
INSERT 0 1
gaussdb=# ANALYZE t1;
ANALYZE
gaussdb=# INSERT INTO t1 VALUES(2,2);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES(2,2);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES(2,2);
INSERT 0 1
gaussdb=# ANALYZE t1;
ANALYZE

-- View the statistics in the history table.
gaussdb=# SELECT staattnum,stadistinct,stakind1,stanumbers1,stavalues1 FROM
GS_STATISTIC_HISTORY WHERE starelid='t1'::REGCLASS ORDER BY statimestamp;
 staattnum | stadistinct | stakind1 | stanumbers1 | stavalues1
-----------+-------------+----------+-------------+------------
         1 |    -.333333 |        1 | {1}         | {1}
         2 |    -.333333 |        1 | {1}         | {1}
         1 |    -.333333 |        1 | {.5,.5}     | {1,2}
         2 |    -.333333 |        1 | {.5,.5}     | {1,2}
(4 rows)

-- Query statistics in the current system catalog.
gaussdb=# SELECT staattnum,stadistinct,stakind1,stanumbers1,stavalues1 FROM PG_STATISTIC
WHERE starelid='t1'::REGCLASS;
 staattnum | stadistinct | stakind1 | stanumbers1 | stavalues1
-----------+-------------+----------+-------------+------------
         1 |    -.333333 |        1 | {.5,.5}     | {1,2}
         2 |    -.333333 |        1 | {.5,.5}     | {1,2}
(2 rows)

-- Roll back to the earlier time point and query the statistics in the system catalog.
gaussdb=# CALL
DBE_STATS.RESTORE_COLUMN_STATS(ownname=>'dbe_stats_restore',tabname=>'t1',colname=>'a',as_
of_timestamp=>((SELECT MIN(reltimestamp) FROM GS_TABLESTATS_HISTORY WHERE relname='t1')
+ INTERVAL '1 second'));
 restore_column_stats
----------------------

(1 row)

gaussdb=# SELECT staattnum,stadistinct,stakind1,stanumbers1,stavalues1 FROM PG_STATISTIC
WHERE starelid='t1'::REGCLASS;
 staattnum | stadistinct | stakind1 | stanumbers1 | stavalues1
-----------+-------------+----------+-------------+------------
         1 |    -.333333 |        1 | {1}         | {1}
         2 |    -.333333 |        1 | {.5,.5}     | {1,2}
(2 rows)

-- Delete a table or namespace.
gaussdb=# DROP TABLE t1;
DROP TABLE
gaussdb=# DROP SCHEMA dbe_stats_restore;
DROP SCHEMA
```

- DBE_STATS.RESTORE_SCHEMA_STATS

  Rolls back statistics of all tables in a specified schema to a specified time point. This API does not return any value.

The prototype of the DBE_STATS.RESTORE_SCHEMA_STATS function is as follows:

```
DBE_STATS.RESTORE_SCHEMA_STATS(
   ownname                VARCHAR2,
   as_of_timestamp          TIMESTAMPTZ,
   force                  BOOLEAN DEFAULT FALSE,
   no_invalidate            BOOLEAN DEFAULT NULL
);
```

Table 10-380 DBE_STATS.RESTORE_SCHEMA_STATS API

Parameter	Type	Whether NULL Is Allowed	Description
ownname	VARCHAR2	Yes	Name of a schema.
as_of_timestamp	timestamp with time zone	No	Target time point to be rolled back.
force	BOOLEAN	Yes	Determines whether to forcibly roll back the statistics when the statistics are locked.
no_invalidate	BOOLEAN	Yes	Not supported currently.

NOTE

- When this API is called, all tables on which the current user has the ANALYZE permission in the schema are rolled back. Tables on which the current user does not have the ANALYZE permission are skipped.
- To call this API, you must have the USAGE permission on the specified schema.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- Roll back schema statistics, which is the same as rolling back table statistics.
-- Create a table, insert data, and collect statistics twice.
gaussdb=#CREATE SCHEMA dbe_stats_restore;
CREATE SCHEMA
gaussdb=#SET CURRENT_SCHEMA=dbe_stats_restore;
SET
gaussdb=#CREATE TABLE t1(a int, b int);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(1,1);
INSERT 0 1
gaussdb=#INSERT INTO t1 VALUES(1,1);
INSERT 0 1
gaussdb=#INSERT INTO t1 VALUES(1,1);
INSERT 0 1
gaussdb=#ANALYZE t1;
ANALYZE
```



```
gaussdb=#INSERT INTO t1 VALUES(2,2);
INSERT 0 1
gaussdb=#INSERT INTO t1 VALUES(2,2);
INSERT 0 1
gaussdb=#INSERT INTO t1 VALUES(2,2);
INSERT 0 1
gaussdb=#ANALYZE t1;
ANALYZE

-- View the system catalog.
gaussdb=#SELECT relname,reltuples FROM GS_TABLESTATS_HISTORY WHERE relname='t1';
 relname | reltuples
---------+-----------
 t1      |         3
 t1      |         6
(2 rows

gaussdb=#SELECT relname,reltuples FROM PG_CLASS WHERE relname='t1' AND relnamespace =
(SELECT oid FROM PG_NAMESPACE WHERE nspname='dbe_stats_restore');
 relname | reltuples
---------+-----------
 t1      |         6
(1 row)

-- Perform rollback.
gaussdb=#CALL
DBE_STATS.RESTORE_SCHEMA_STATS(ownname=>'dbe_stats_restore',as_of_timestamp=>((SELECT
MIN(reltimestamp) FROM GS_TABLESTATS_HISTORY WHERE relname='t1') + INTERVAL '1 second'));
 restore_schema_stats
----------------------

(1 row)

-- Check the system catalog again.
gaussdb=#SELECT relname,reltuples FROM PG_CLASS WHERE relname='t1' AND relnamespace =
(SELECT oid FROM PG_NAMESPACE WHERE nspname='dbe_stats_restore');
 relname | reltuples
---------+-----------
 t1      |         3
(1 row)

-- Delete a table or namespace.
gaussdb=# DROP TABLE t1;
DROP TABLE
gaussdb=# DROP SCHEMA dbe_stats_restore;
DROP SCHEMA
```

- DBE_STATS.PURGE_STATS

  Clears all historical statistics before a specified time point. This API does not return any value.

  The prototype of the DBE_STATS.PURGE_STATS function is as follows:

```
DBE_STATS.PURGE_STATS(
   before_timestamp       TIMESTAMPTZ
);
```

Table 10-381 DBE_STATS.PURGE_STATS API

Parameter	Type	Whether NULL Is Allowed	Description
before_timestamp	timestamp with time zone	No	Time node for clearing.

NOTE

Only the initial user can call this API.

Example:

```
-- Create a table and collect statistics twice.
gaussdb=# CREATE SCHEMA dbe_stats_purge;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA=dbe_stats_purge;
SET
gaussdb=# CREATE TABLE t1(a int, b int);
CREATE TABLE
gaussdb=# INSERT INTO t1 VALUES(1,1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES(1,1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES(1,1);
INSERT 0 1
gaussdb=# ANALYZE t1;
ANALYZE
gaussdb=# INSERT INTO t1 VALUES(2,2);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES(2,2);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES(2,2);
INSERT 0 1
gaussdb=# ANALYZE t1;
ANALYZE

-- View history tables.
gaussdb=# SELECT relname,reltuples FROM GS_TABLESTATS_HISTORY WHERE relname='t1';
 relname | reltuples
---------+-----------
 t1      |         3
 t1      |         6
(2 rows)

-- Clear the earlier history statistics and view the history table.
gaussdb=# CALL DBE_STATS.PURGE_STATS(before_timestamp=>((SELECT MIN(reltimestamp) FROM
GS_TABLESTATS_HISTORY WHERE relname='t1') + INTERVAL '1 second'));
 purge_stats
-------------


(1 row)

gaussdb=# SELECT relname,reltuples FROM GS_TABLESTATS_HISTORY WHERE relname='t1';
 relname | reltuples
---------+-----------
 t1      |         6
(1 row)

-- Delete a table or namespace.
gaussdb=# DROP TABLE t1;
DROP TABLE
gaussdb=# DROP SCHEMA dbe_stats_purge;
DROP SCHEMA
```

- DBE_STATS.GET_STATS_HISTORY_RETENTION

  Obtains the retention period of historical statistics. The retention period set by DBE_STATS.ALTER_STATS_HISTORY_RETENTION is returned. This API does not have input parameters.

  The prototype of the DBE_STATS.GET_STATS_HISTORY_RETENTION function is as follows:

  ```
  DBE_STATS.GET_STATS_HISTORY_RETENTION()
  returns NUMBER;
  ```

  Example:

```
gaussdb=# CALL DBE_STATS.GET_STATS_HISTORY_RETENTION();
 get_stats_history_retention
-----------------------------
                          31
(1 row)
```

- DBE_STATS.GET_STATS_HISTORY_AVAILABILITY

  Obtains the earliest available time of the current historical statistics. The earliest time of all historical statistics in the current database is returned. This API does not have input parameters.

  The prototype of the DBE_STATS.GET_STATS_HISTORY_AVAILABILITY function is as follows:

```
DBE_STATS.GET_STATS_HISTORY_AVAILABILITY()
returns TIMESTAMPTZ;
```

  Example:

```
gaussdb=# ANALYZE;
ANALYZE
gaussdb=# CALL DBE_STATS.GET_STATS_HISTORY_AVAILABILITY();
 get_stats_history_availability
--------------------------------
 2023-08-27 02:07:04.065217+08
(1 row)
```

  NOTE

  All the preceding function prototypes are the C_FUNCTION function prototypes with the same data types and columns, and are named **DBE_STATS.***XXXX*_**C_FUNCTION**.

  The DBE_STATS advanced package API is valid only for ordinary tables.

- DBE_STATS.CREATE_STAT_TABLE

  Creates a statistics table for storing user statistics. This API does not return any value.

  The prototype of the DBE_STATS.CREATE_STAT_TABLE function is as follows:

```
DBE_STATS.CREATE_STAT_TABLE (
   ownname          VARCHAR2,
   stattab          VARCHAR2,
   tblspace         VARCHAR2 DEFAULT NULL,
   global_temporary BOOLEAN DEFAULT NULL
);
```

Table 10-382 DBE_STATS.CREATE_STAT_TABLE parameters

Parameter	Type	Whether NULL Is Allowed	Description
ownname	VARCHAR2	No	Schema where the created statistics table is located. Null indicates that the schema is created in the current schema by default.
stattab	VARCHAR2	No	Name of a statistics table.

Parameter	Type	Whether NULL Is Allowed	Description
tblspace	VARCHAR2	Yes	Tablespace where the created statistics table is located. If not specified, the default tablespace is used.
global_temporary	BOOLEAN	Yes	This parameter is not supported currently.

NOTE

- To call this procedure to create a statistics table, you must have the permission to create tables and indexes in the specified schema and tablespace.
- To distinguish ordinary tables from statistics tables, the **statstable** column in **reloptions** of a statistics table is not an empty string after the statistics table is created. Considering that the structure of the statistics table may be upgraded in the future, to implement forward compatibility, **statstable** is assigned the version information of the current statistics table.
- When this table is created, the gs_stattab_*oid*_statid_type_nameinfo_index index is automatically generated for index scanning. *oid* is the OID of the statistics table. If the index is deleted, the export and import APIs are unavailable.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- Create a stattab statistics table in dbe_stats_create_drop_schema and manually insert a data record. You can query the data record in the new user statistics table.
gaussdb=# CREATE SCHEMA dbe_stats_create_drop_schema;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA = dbe_stats_create_drop_schema;
SET
gaussdb=# CALL DBE_STATS.CREATE_STAT_TABLE(ownname => 'dbe_stats_create_drop_schema', stattab => 'stat_tab');
 create_stat_table
-------------------
(1 row)
gaussdb=# INSERT INTO stat_tab(statid, type, relname) VALUES('123', 't', 't1');
INSERT 0 1
gaussdb=# SELECT statid, type, relname FROM stat_tab;
 statid | type | relname
---------+------+---------
 123    | t    | t1
(1 row)
```

- DBE_STATS.DROP_STAT_TABLE

  Deletes the user statistics table. This API does not return any value.

  The prototype of the DBE_STATS.DROP_STAT_TABLE function is as follows:

```
DBE_STATS.DROP_STAT_TABLE (
   ownname VARCHAR2,
   stattab VARCHAR2
);
```


Table 10-383 DBE_STATS.DROP_STAT_TABLE parameters

Parameter	Type	Whether NULL Is Allowed	Description
ownname	VARCHAR2	No	Schema where the statistics table is located. Null indicates that the current schema is used by default.
stattab	VARCHAR2	No	Name of a statistics table.

NOTE

- To call this procedure to delete a statistics table, you must have the permission to delete the table and index in the specified schema.
- Ordinary tables whose **statstable** attribute is empty cannot be deleted using this procedure.
- If an index is created on the user statistics table or other objects that depend on the table are created, all objects that depend on the table are deleted when this procedure is called.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- Delete the created user statistics table and the new schema.
gaussdb=# CALL DBE_STATS.DROP_STAT_TABLE(ownname => 'dbe_stats_create_drop_schema',
stattab => 'stat_tab');
 drop_stat_table
-----------------
(1 row)
gaussdb=# DROP SCHEMA dbe_stats_create_drop_schema CASCADE;
DROP SCHEMA
```

- DBE_STATS.EXPORT_INDEX_STATS

  Searches the system catalog for the statistics of the specified index according to and store the result in the user statistics table. This API does not return any value.

  The prototype of the DBE_STATS.EXPORT_INDEX_STATS function is:

```
DBE_STATS.EXPORT_INDEX_STATS (
ownname  VARCHAR2,
indname  VARCHAR2,
partname VARCHAR2 DEFAULT NULL,
stattab  VARCHAR2,
statid   VARCHAR2 DEFAULT NULL,
statown  VARCHAR2 DEFAULT NULL
);
```


Table 10-384 DBE_STATS.EXPORT_INDEX_STATS parameters

Parameter	Type	Whether NULL Is Allowed	Description
ownname	VARCHAR2	No	Name of the schema where the index is located. Null indicates that the current schema is used by default.
indname	VARCHAR2	No	Name of the index whose statistics are to be exported.
partname	VARCHAR2	Yes	Name of a partitioned index. If the index has been partitioned but **partname** is **null**, global and partitioned index statistics are exported.
stattab	VARCHAR2	No	Name of the user table that stores statistics.
statid	VARCHAR2	Yes	(Optional) Identifier that points to statistics in stattab.
statown	VARCHAR2	Yes	Not supported currently.

NOTE

- To call this procedure to export index statistics, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- To call this procedure to export statistics to a user statistics table, you must have the SELECT, INSERT, DELETE, and UPDATE permissions on the user statistics table.
- You need to export the reltuples (numrows), relpages (numlblks), and relallvisible (relallvisible) statistics of indexes.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- Create a table and collect statistics.
gaussdb=# CREATE SCHEMA dbe_stats_export;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA = dbe_stats_export;
SET
gaussdb=# CREATE TABLE t2(a int, b int, c int) PARTITION BY RANGE(a) SUBPARTITION BY RANGE(b)
 (
   PARTITION p1 VALUES LESS THAN(200)
   (SUBPARTITION s1 VALUES LESS THAN (500),
    SUBPARTITION s2 VALUES LESS THAN (MAXVALUE)
   ),
   PARTITION p2 VALUES LESS THAN(500)
   (SUBPARTITION s3 VALUES LESS THAN (500),
    SUBPARTITION s4 VALUES LESS THAN (MAXVALUE)
   ),
   PARTITION p3 VALUES LESS THAN(MAXVALUE)
   ( SUBPARTITION s5 VALUES LESS THAN (500),
     SUBPARTITION s6 VALUES LESS THAN (MAXVALUE)
   )
 );
CREATE TABLE
gaussdb=# CREATE INDEX idx_t2_local_a ON t2(a) LOCAL
 (
    PARTITION p1_idx_a
    (
       SUBPARTITION s1_idx_a,
       SUBPARTITION s2_idx_a
    ),
    PARTITION p2_idx_a
    (
       SUBPARTITION s3_idx_a,
       SUBPARTITION s4_idx_a
    ),
    PARTITION p3_idx_a
    (
       SUBPARTITION s5_idx_a,
       SUBPARTITION s6_idx_a
    ));
CREATE INDEX
gaussdb=# INSERT INTO t2 VALUES (100, 300, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (100, 600, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (300, 300, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (300, 600, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (600, 400, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (600, 600, 1);
INSERT 0 1
gaussdb=# ANALYZE t2;
ANALYZE
gaussdb=# ANALYZE t2((a,b));
ANALYZE
-- Create a user statistics table as the target table for exporting statistics.
gaussdb=# CALL DBE_STATS.CREATE_STAT_TABLE('dbe_stats_export', 'export_table');
 create_stat_table
-------------------
(1 row)

-- Export index statistics together with partitioned index statistics.
-- View the current statistics.
gaussdb=# SELECT relpages,reltuples FROM PG_CLASS WHERE relname='idx_t2_local_a';
 relpages | reltuples
----------+-----------
        6 |         6
(1 row)
```



```
gaussdb=# SELECT relname,relpages,reltuples FROM PG_PARTITION WHERE
parentid='idx_t2_local_a'::REGCLASS;
 relname  | relpages | reltuples
----------+----------+-----------
 s1_idx_a |        1 |         0
 s2_idx_a |        1 |         0
 s3_idx_a |        1 |         0
 s4_idx_a |        1 |         0
 s5_idx_a |        1 |         0
 s6_idx_a |        1 |         0
(6 rows)
-- Export the statistics and view the data in the user statistics table.
gaussdb=# CALL DBE_STATS.EXPORT_INDEX_STATS(ownname => 'dbe_stats_export', indname =>
'idx_t2_local_a', partname=> null, stattab => 'export_table', statid => 'idx_s1');
 export_index_stats
--------------------
(1 row)
gaussdb=# SELECT relname,partname,numrows,numblocks FROM export_table;
    relname     | partname | numrows | numblocks
----------------+----------+---------+-----------
 idx_t2_local_a |          |       6 |         6
 idx_t2_local_a | s6_idx_a |       0 |         1
 idx_t2_local_a | s5_idx_a |       0 |         1
 idx_t2_local_a | s4_idx_a |       0 |         1
 idx_t2_local_a | s3_idx_a |       0 |         1
 idx_t2_local_a | s2_idx_a |       0 |         1
 idx_t2_local_a | s1_idx_a |       0 |         1
(7 rows)
-- Delete a table or namespace.
gaussdb=# DROP INDEX idx_t2_local_a;
DROP INDEX
gaussdb=# DROP TABLE t2;
DROP TABLE
gaussdb=# DROP TABLE export_table;
DROP TABLE
gaussdb=# DROP SCHEMA dbe_stats_export;
DROP SCHEMA
```

- DBE_STATS.EXPORT_TABLE_STATS

  Retrieves statistics for the specified table and stores them in the user statistics table. This API does not return any value.

  The prototype of the DBE_STATS.EXPORT_TABLE_STATS function is as follows:

```
DBE_STATS.EXPORT_TABLE_STATS (
  ownname          VARCHAR2,
  tabname          VARCHAR2,
  partname         VARCHAR2 DEFAULT NULL,
  stattab          VARCHAR2,
  statid           VARCHAR2 DEFAULT NULL,
  cascade          BOOLEAN  DEFAULT TRUE,
  statown          VARCHAR2 DEFAULT NULL,
  stat_category    VARCHAR2 DEFAULT NULL
);
```

Table 10-385 DBE_STATS.EXPORT_TABLE_STATS parameters

Parameter	Type	Whether NULL Is Allowed	Description
ownname	VARCHAR2	No	Name of the schema. Null indicates that the current schema is used by default.

Parameter	Type	Whether NULL Is Allowed	Description
tabname	VARCHAR2	No	Name of the table whose statistics are to be exported.
partname	VARCHAR2	Yes	Table partition name. If the table has been partitioned but **partname** is **null**, global and partitioned table statistics are exported.
stattab	VARCHAR2	No	Name of the user table that stores statistics.
statid	VARCHAR2	Yes	(Optional) Identifier that points to statistics in stattab.
cascade	BOOLEAN	Yes	Specifies whether to export statistics about columns, indexes, multiple columns, and index expressions. The default value is **TRUE**.
statown	VARCHAR2	Yes	Not supported currently.
stat_category	VARCHAR2	Yes	Not supported currently.

NOTE

- To call this procedure to export table statistics, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- To call this procedure to export statistics to a user statistics table, you must have the SELECT, INSERT, DELETE, and UPDATE permissions on the user statistics table.
- You need to export the reltuples (numrows), relpages (numblks), and relallvisible (relallvisible) statistics of tables. For details about how to export column-level and index-level statistics in cascading mode, see the statistics of the **EXPORT_INDEX_STATS** and **EXPORT_COLUMN_STATS** APIs.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- The method of exporting table statistics is the same as that of exporting index statistics. The following describes only the API calling method. For details, see the DBE_STATS.EXPORT_INDEX_STATS example.
gaussdb=# CALL DBE_STATS.EXPORT_TABLE_STATS(ownname => 'dbe_stats_export', tabname => 't2', partname =>null, stattab => 'export_table', statid => 't1_t', cascade => true);
 export_table_stats
--------------------
(1 row)
```

- DBE_STATS.EXPORT_COLUMN_STATS

  Retrieves statistics for the specified columns (multiple columns or expressions) and stores them in the user statistics table. This API does not return any value.

  The prototype of the DBE_STATS.EXPORT_COLUMN_STATS function is:

```
DBE_STATS.EXPORT_COLUMN_STATS (
   ownname  VARCHAR2,
   tabname  VARCHAR2,
   colname  VARCHAR2,
   partname VARCHAR2 DEFAULT NULL,
   stattab  VARCHAR2,
   statid   VARCHAR2 DEFAULT NULL,
   statown  VARCHAR2 DEFAULT NULL
);
```

Table 10-386 DBE_STATS.EXPORT_COLUMN_STATS parameters

Parameter	Type	Whether NULL Is Allowed	Description
ownname	VARCHAR2	No	Name of the schema. Null indicates that the current schema is used by default.

Parameter	Type	Whether NULL Is Allowed	Description
tabname	VARCHAR2	No	Name of the table (index name corresponding to the expression) to which the column whose statistics are to be exported belongs.
colname	VARCHAR2	No	Column name or multiple column names (If tabname is an index, **colname** is an expression name.)
partname	VARCHAR2	Yes	Partition name. If the table or index has been partitioned but **partname** is **null**, global and partitioned key, multiple columns, and index statistics are exported.
stattab	VARCHAR2	No	Name of the user table that stores statistics.
statid	VARCHAR2	Yes	(Optional) Identifier that points to statistics in stattab.
statown	VARCHAR2	Yes	Not supported currently.

NOTE

- To call this procedure to export table statistics, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- To call this procedure to export statistics to a user statistics table, you must have the SELECT, INSERT, DELETE, and UPDATE permissions on the user statistics table.
- You can query **PG_STATISTIC** or **PG_STATISTIC_EXT** to export column-level statistics.
- Before exporting the statistics of multiple columns, query the **staextname** column in **PG_STATISTIC_EXT** to obtain the aliases of multiple columns and transfer them as the input parameter **colname**.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- Create a table and collect statistics.
gaussdb=# CREATE SCHEMA dbe_stats_export;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA=dbe_stats_export;
SET
gaussdb=# CREATE TABLE IF NOT EXISTS t1 (a int, b int, c int);
CREATE TABLE
gaussdb=# CREATE INDEX idx_t1_expr on t1((a*a), (a+a), (a*b));
CREATE INDEX
gaussdb=# INSERT INTO t1 VALUES (100, 300, 1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (150, 550, 1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (250, 150, 1);
INSERT 0 1
gaussdb=# ANALYZE t1;
ANALYZE
gaussdb=# ANALYZE t1((a,b));
ANALYZE
-- Create a user statistics table.
gaussdb=# CALL DBE_STATS.CREATE_STAT_TABLE('dbe_stats_export', 'export_table');
 create_stat_table
-------------------
(1 row)

-- View column-level statistics.
gaussdb=# SELECT staattnum,stadistinct,stakind1,stanumbers1,stavalues1 FROM PG_STATISTIC
WHERE starelid='t1'::REGCLASS;
 staattnum | stadistinct | stakind1 | stanumbers1 |  stavalues1
-----------+-------------+----------+-------------+--------------
         1 |          -1 |        2 |             | {100,150,250}
         2 |          -1 |        2 |             | {150,300,550}
         3 |    -.333333 |        1 | {1}         | {1}
(3 rows)
-- Export column-level statistics and view the user statistics table.
gaussdb=# CALL DBE_STATS.EXPORT_COLUMN_STATS(ownname => 'dbe_stats_export', tabname =>
't1', colname=> 'a', partname => null, stattab => 'export_table', statid => 't1_c');
 export_column_stats
---------------------
(1 row)
gaussdb=# SELECT relname,columnname,stadistinct,stakind1,stanumbers1,stavalues1 FROM
export_table;
 relname | columnname | stadistinct | stakind1 | stanumbers1 |  stavalues1
---------+------------+-------------+----------+-------------+--------------
 t1      | a          |          -1 |        2 |             | {100,150,250}
(1 row)

-- View expression statistics.
gaussdb=# SELECT staattnum,stadistinct,stakind1,stanumbers1,stavalues1 FROM PG_STATISTIC
WHERE starelid='idx_t1_expr'::REGCLASS;
```



```
 staattnum | stadistinct | stakind1 | stanumbers1 |     stavalues1
-----------+-------------+----------+-------------+---------------------
         1 |          -1 |        2 |             | {10000,22500,62500}
         2 |          -1 |        2 |             | {200,300,500}
         3 |          -1 |        2 |             | {30000,37500,82500}
(3 rows)
-- Export expression statistics and view the user statistics table.
gaussdb=# CALL DBE_STATS.EXPORT_COLUMN_STATS(ownname => 'dbe_stats_export', tabname =>
'idx_t1_expr', colname => 'expr', stattab => 'export_table', statid => 't1_expr');
 export_column_stats
---------------------
(1 row)
gaussdb=# SELECT relname,columnname,stadistinct,stakind1,stanumbers1,stavalues1 FROM
export_table where statid='t1_expr';
   relname    | columnname | stadistinct | stakind1 | stanumbers1 |     stavalues1

--------------+------------+-------------+----------+-------------+---------------------
 idx_t1_expr | expr       |          -1 |        2 |             | {10000,22500,62500}
(1 row)

-- Export the statistics of multiple columns.
-- View the statistics of multiple columns.
gaussdb=# SELECT stakey,staextname,stadistinct,stakind1,stanumbers1,stavalues1 FROM
PG_STATISTIC_EXT WHERE starelid='t1'::REGCLASS;
 stakey |     staextname      | stadistinct | stakind1 | stanumbers1 | stavalues1
--------+---------------------+-------------+----------+-------------+------------
 1 2    | extname_-1308656218 |          -1 |        8 |             |
(1 row)
-- Export and view the user statistics table.
gaussdb=# CALL DBE_STATS.EXPORT_COLUMN_STATS(ownname => 'dbe_stats_export', tabname =>
't1', colname=> 'extname_-1308656218', partname => null, stattab => 'export_table', statid => 't1_c');
 export_column_stats
---------------------
(1 row)
gaussdb=# SELECT relname,columnname,stadistinct,stakind1,stanumbers1,stavalues1 FROM
export_table;
 relname |     columnname      | stadistinct | stakind1 | stanumbers1 |  stavalues1
---------+---------------------+-------------+----------+-------------+---------------
 t1      | a                   |          -1 |        2 |             | {100,150,250}
 t1      | extname_-1308656218 |          -1 |        8 |             |
(2 rows)
-- Delete a table or namespace.
gaussdb=# DROP TABLE t1;
DROP TABLE
gaussdb=# DROP TABLE export_table;
DROP TABLE
gaussdb=# DROP SCHEMA dbe_stats_export;
DROP SCHEMA
```

- DBE_STATS.EXPORT_SCHEMA_STATS

  Retrieves statistics for the specified schema and stores them in the user statistics table. This API does not return any value.

  The prototype of the DBE_STATS.EXPORT_SCHEMA_STATS function is:

```
DBE_STATS.EXPORT_TABLE_STATS (
  ownname         VARCHAR2,
  stattab         VARCHAR2,
  statid          VARCHAR2 DEFAULT NULL,
  statown         VARCHAR2 DEFAULT NULL,
  stat_category   VARCHAR2 DEFAULT NULL);
```


Table 10-387 DBE_STATS.EXPORT_SCHEMA_STATS parameters

Parameter	Type	Whether NULL Is Allowed	Description
ownname	VARCHAR2	No	Name of the schema. Null indicates that the current schema is used by default.
stattab	VARCHAR2	No	Name of the user table that stores statistics.
statid	VARCHAR2	Yes	(Optional) Identifier that points to statistics in stattab.
statown	VARCHAR2	Yes	Not supported currently.
stat_category	VARCHAR2	Yes	Not supported currently.

NOTE

- To call this procedure to export table statistics, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- To call this procedure to export statistics to a user statistics table, you must have the SELECT, INSERT, DELETE, and UPDATE permissions on the user statistics table.
- You need to export statistics about all tables on which the current user has permission in the specified schema. Tables on which the current user does not have permission are skipped and no error is reported.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- The method of exporting schema statistics is the same as that of exporting table statistics. The following describes only the API calling method. For details, see the DBE_STATS.EXPORT_TABLE_STATS example.
gaussdb=# CALL DBE_STATS.EXPORT_SCHEMA_STATS(ownname => 'dbe_stats_export', stattab => 'export_table', statid => 's1');
 export_schema_stats
---------------------
(1 row)
```

- DBE_STATS.IMPORT_INDEX_STATS

  Retrieves statistics for the specified index from the user statistics table of the stattab identifier and writes it back to the system catalog. This API does not return any value.

  The prototype of the DBE_STATS.IMPORT_INDEX_STATS function is as follows:


```
DBE_STATS.IMPORT_INDEX_STATS (
   ownname        VARCHAR2,
   indname        VARCHAR2,
   partname       VARCHAR2 DEFAULT NULL,
   stattab        VARCHAR2,
   statid         VARCHAR2 DEFAULT NULL,
   statown        VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT NULL,
   force          BOOLEAN DEFAULT FALSE
);
```

Table 10-388 DBE_STATS.IMPORT_INDEX_STATS parameters

Parameter	Type	Whether NULL Is Allowed	Description
ownname	VARCHAR2	No	Name of the schema. Null indicates that the current schema is used by default.
indname	VARCHAR2	No	Name of the index whose statistics are to be imported.
partname	VARCHAR2	Yes	Name of the index partition. If the index has been partitioned but **partname** is **null**, global and partitioned index statistics are imported.
stattab	VARCHAR2	No	Name of the user table that stores statistics.
statid	VARCHAR2	Yes	(Optional) Identifier that points to statistics in stattab.
statown	VARCHAR2	Yes	Not supported currently.
no_invalidate	BOOLEAN	Yes	Not supported currently.

Parameter	Type	Whether NULL Is Allowed	Description
force	BOOLEAN	Yes	Specifies whether to forcibly import data. The lock status is ignored. The default value is **FALSE**.

NOTE

- To call this procedure to import statistics, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- To call this procedure to import statistics from a user statistics table to a system catalog, you must have the SELECT, INSERT, DELETE, and UPDATE permissions on the user statistics table.
- You need to import the reltuples (numrows), relpages (numlblks), and relallvisible (relallvisible) statistics of indexes.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- Create a table and collect statistics.
gaussdb=# CREATE SCHEMA dbe_stats_import;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA = dbe_stats_import;
SET
gaussdb=# CREATE TABLE t2(a int, b int, c int) PARTITION BY RANGE(a) SUBPARTITION BY RANGE(b)
 (
   PARTITION p1 VALUES LESS THAN(200)
   (SUBPARTITION s1 VALUES LESS THAN (500),
    SUBPARTITION s2 VALUES LESS THAN (MAXVALUE)
   ),
   PARTITION p2 VALUES LESS THAN(500)
   (SUBPARTITION s3 VALUES LESS THAN (500),
    SUBPARTITION s4 VALUES LESS THAN (MAXVALUE)
   ),
   PARTITION p3 VALUES LESS THAN(MAXVALUE)
   ( SUBPARTITION s5 VALUES LESS THAN (500),
     SUBPARTITION s6 VALUES LESS THAN (MAXVALUE)
   )
 );
CREATE TABLE
gaussdb=# CREATE INDEX idx_t2_local_a ON t2(a) LOCAL
 (
    PARTITION p1_idx_a
    (
       SUBPARTITION s1_idx_a,
       SUBPARTITION s2_idx_a
    ),
    PARTITION p2_idx_a
    (
       SUBPARTITION s3_idx_a,
       SUBPARTITION s4_idx_a
    ),
    PARTITION p3_idx_a
    (
```



```
        SUBPARTITION s5_idx_a,
        SUBPARTITION s6_idx_a
    ));
CREATE INDEX
gaussdb=# INSERT INTO t2 VALUES (100, 300, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (100, 600, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (300, 300, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (300, 600, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (600, 400, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (600, 600, 1);
INSERT 0 1
gaussdb=# ANALYZE t2;
ANALYZE
-- Create a user statistics table and export the data.
gaussdb=# CALL DBE_STATS.CREATE_STAT_TABLE('dbe_stats_import', 'export_table');
 create_stat_table
-------------------
(1 row)
gaussdb=# CALL DBE_STATS.IMPORT_INDEX_STATS(ownname => 'dbe_stats_import', indname =>
'idx_t2_local_a', partname=> null, stattab => 'export_table', statid => 'idx_s1');
 export_index_stats
--------------------
(1 row)
gaussdb=# SELECT relname,partname,numrows,numblocks FROM export_table;
    relname     | partname | numrows | numblocks
----------------+----------+---------+-----------
 idx_t2_local_a |          |       6 |         6
 idx_t2_local_a | s6_idx_a |       0 |         1
 idx_t2_local_a | s5_idx_a |       0 |         1
 idx_t2_local_a | s4_idx_a |       0 |         1
 idx_t2_local_a | s3_idx_a |       0 |         1
 idx_t2_local_a | s2_idx_a |       0 |         1
 idx_t2_local_a | s1_idx_a |       0 |         1
(7 rows)
-- Insert data again, collect statistics, and view the statistics.
gaussdb=# INSERT INTO t2 VALUES (100, 300, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (100, 600, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (100, 600, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (300, 600, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (600, 400, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (600, 600, 1);
INSERT 0 1
gaussdb=# ANALYZE t2;
ANALYZE
gaussdb=# SELECT relpages,reltuples FROM PG_CLASS WHERE relname='idx_t2_local_a';
 relpages | reltuples
----------+-----------
        6 |        12
(1 row)
gaussdb=# SELECT relname,relpages,reltuples FROM PG_PARTITION WHERE
parentid='idx_t2_local_a'::REGCLASS;
 relname  | relpages | reltuples
----------+----------+-----------
 s1_idx_a |        1 |         0
 s2_idx_a |        1 |         0
 s3_idx_a |        1 |         0
 s4_idx_a |        1 |         0
 s5_idx_a |        1 |         0
 s6_idx_a |        1 |         0
```



```
(6 rows)
-- Import the statistics in the user statistics table to the statistics information system table.
gaussdb=# CALL DBE_STATS.IMPORT_INDEX_STATS(ownname => 'dbe_stats_import', indname =>
'idx_t2_local_a', stattab => 'export_table', statid => 'idx_s1');
 import_index_stats
--------------------

(1 row)
gaussdb=# SELECT relpages,reltuples FROM PG_CLASS WHERE relname='idx_t2_local_a';
 relpages | reltuples
----------+-----------
        6 |         6
(1 row)
gaussdb=# SELECT relname,relpages,reltuples FROM PG_PARTITION WHERE
parentid='idx_t2_local_a'::REGCLASS;
 relname  | relpages | reltuples
----------+----------+-----------
 s1_idx_a |        1 |         0
 s2_idx_a |        1 |         0
 s3_idx_a |        1 |         0
 s4_idx_a |        1 |         0
 s5_idx_a |        1 |         0
 s6_idx_a |        1 |         0
(6 rows)
-- Delete a table or namespace.
gaussdb=# DROP TABLE t2;
DROP TABLE
gaussdb=# DROP TABLE export_table;
DROP TABLE
gaussdb=# DROP SCHEMA dbe_stats_import;
DROP SCHEMA
```

- DBE_STATS.IMPORT_TABLE_STATS

  Retrieves statistics for the specified table from the user statistics table of the stattab identifier and writes it back to the system catalog. This API does not return any value.

  The prototype of the DBE_STATS.IMPORT_TABLE_STATS function is as follows:

```
DBE_STATS.IMPORT_TABLE_STATS (
   ownname       VARCHAR2,
   tabname       VARCHAR2,
   partname      VARCHAR2 DEFAULT NULL,
   stattab       VARCHAR2,
   statid        VARCHAR2 DEFAULT NULL,
   cascade       BOOLEAN  DEFAULT TRUE,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN  DEFAULT NULL,
   force         BOOLEAN  DEFAULT FALSE,
   stat_category VARCHAR2 DEFAULT NULL
);
```

Table 10-389 DBE_STATS.IMPORT_TABLE_STATS parameters

Parameter	Type	Whether NULL Is Allowed	Description
ownname	varchar2	No	Name of the schema. Null indicates that the current schema is used by default.

Parameter	Type	Whether NULL Is Allowed	Description
tabname	varchar2	No	Name of the table whose statistics are to be imported.
partname	varchar2	Yes	Table partition name. If the table has been partitioned but **partname** is **null**, global and partitioned table statistics are imported.
stattab	varchar2	No	Name of the user table that stores statistics.
statid	varchar2	Yes	(Optional) Identifier that points to statistics in stattab.
cascade	Boolean	Yes	Specifies whether to import statistics about columns, indexes, multiple columns, and index expressions. The default value is **true**.
statown	varchar2	Yes	Not supported currently.
no_invalidate	Boolean	Yes	Not supported currently.
force	Boolean	Yes	Specifies whether to forcibly import data. The lock status is ignored. The default value is **FALSE**.
stat_category	varchar2	Yes	Not supported currently.

NOTE

- To call this procedure to import statistics, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- To call this procedure to import statistics from a user statistics table to a system catalog, you must have the SELECT, INSERT, DELETE, and UPDATE permissions on the user statistics table.
- You need to import the reltuples (numrows), relpages (numblks), and relallvisible (relallvisible) statistics of tables. For details about column-level and index-level statistics imported in cascading mode, see the statistics of the **IMPORT_INDEX_STATS** and **IMPORT_COLUMN_STATS** APIs.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- The method of importing table statistics is the same as that of importing index statistics. The following describes only the API calling method. For details, see the DBE_STATS.IMPORT_INDEX_STATS example.
gaussdb=# CALL DBE_STATS.IMPORT_TABLE_STATS(ownname => 'dbe_stats_import', tabname => 't2', partname => 'p1', stattab => 'export_table', statid => 't1_t', cascade => true, force => false);
 import_table_stats
--------------------
(1 row)
```

- DBE_STATS.IMPORT_COLUMN_STATS

  Retrieves statistics for the specified columns (multiple columns or expressions) from the user statistics table of the stattab identifier and writes it back to the system catalog. This API does not return any value.

  The prototype of the DBE_STATS.IMPORT_COLUMN_STATS function is as follows:

```
DBE_STATS.IMPORT_COLUMN_STATS (
   ownname       VARCHAR2,
   tabname       VARCHAR2,
   colname       VARCHAR2,
   partname      VARCHAR2 DEFAULT NULL,
   stattab       VARCHAR2,
   statid        VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN DEFAULT NULL,
   force         BOOLEAN DEFAULT FALSE
);
```

Table 10-390 DBE_STATS.IMPORT_COLUMN_STATS parameters

Parameter	Type	Whether NULL Is Allowed	Description
ownname	VARCHAR2	No	Name of the schema. Null indicates that the current schema is used by default.

Parameter	Type	Whether NULL Is Allowed	Description
tabname	VARCHAR2	No	Name of the table (index name corresponding to the expression) to which the column whose statistics are to be imported belongs.
colname	VARCHAR2	No	Column name or multiple column names (If tabname is an index, **colname** is an expression name.)
partname	VARCHAR2	Yes	Partition name. If the table or index has been partitioned but **partname** is **null**, global and partitioned key, multiple columns, and index statistics are imported.
stattab	VARCHAR2	No	Name of the user table that stores statistics.
statid	VARCHAR2	Yes	(Optional) Identifier that points to statistics in stattab.
statown	VARCHAR2	Yes	Not supported currently.
no_invalidate	BOOLEAN	Yes	Not supported currently.

Parameter	Type	Whether NULL Is Allowed	Description
force	BOOLEAN	Yes	Specifies whether to forcibly import data. The lock status is ignored. The default value is **FALSE**.

NOTE

- To call this procedure to import statistics, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- To call this procedure to import statistics from a user statistics table to a system catalog, you must have the SELECT, INSERT, DELETE, and UPDATE permissions on the user statistics table.
- You can query **PG_STATISTIC** or **PG_STATISTIC_EXT** to import column-level statistics.
- Before importing multiple columns, query the **staextname** column in **PG_STATISTIC_EXT** to obtain the aliases of multiple columns and transfer them as the input parameter **colname**.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- Create a table and collect statistics.
gaussdb=# CREATE SCHEMA dbe_stats_import;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA = dbe_stats_import;
SET
gaussdb=# CREATE TABLE IF NOT EXISTS t1 (a int, b int, c int);
CREATE TABLE
gaussdb=# CREATE INDEX idx_t1_expr on t1((a*a), (a+a), (a*b));
CREATE INDEX
gaussdb=# INSERT INTO t1 VALUES (100, 300, 1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (150, 550, 1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (250, 150, 1);
INSERT 0 1
gaussdb=# ANALYZE t1;
ANALYZE
gaussdb=# ANALYZE t1((a,b));
ANALYZE
-- Create a user statistics table and export the statistics of a single column or multiple columns.
gaussdb=# CALL DBE_STATS.CREATE_STAT_TABLE('dbe_stats_import', 'export_table');
 create_stat_table
-------------------
(1 row)
gaussdb=# SELECT stakey,staextname,stadistinct,stakind1,stanumbers1,stavalues1 FROM
PG_STATISTIC_EXT WHERE starelid='t1'::REGCLASS;
 stakey |     staextname      | stadistinct | stakind1 | stanumbers1 | stavalues1
--------+---------------------+-------------+----------+-------------+------------
 1 2    | extname_-1308656218 |          -1 |        8 |             |
(1 row)
gaussdb=# CALL DBE_STATS.EXPORT_COLUMN_STATS(ownname => 'dbe_stats_import', tabname =>
't1', colname=> 'a', partname => null, stattab => 'export_table', statid => 't1_c');
```



```
 export_column_stats
---------------------
(1 row)
gaussdb=# CALL DBE_STATS.EXPORT_COLUMN_STATS(ownname => 'dbe_stats_import', tabname =>
'idx_t1_expr', colname => 'expr', stattab => 'export_table', statid => 't1_expr');
 export_column_stats
---------------------
(1 row)
gaussdb=# CALL DBE_STATS.EXPORT_COLUMN_STATS(ownname => 'dbe_stats_import', tabname =>
't1', colname=> 'extname_-1308656218', partname => null, stattab => 'export_table', statid => 't1_c');
 export_column_stats
---------------------
(1 row)
gaussdb=# SELECT relname,columnname,stadistinct,stakind1,stanumbers1,stavalues1 FROM
export_table;
   relname   |     columnname      | stadistinct | stakind1 | stanumbers1 |     stavalues1
-------------+---------------------+-------------+----------+-------------+---------------------
 idx_t1_expr | expr                |          -1 |        2 |             | {10000,22500,62500}
 t1          | a                   |          -1 |        2 |             | {100,150,250}
 t1          | extname_-1308656218 |          -1 |        8 |             |
(3 rows)
-- Insert data again, collect statistics, and view the system catalog.
gaussdb=# INSERT INTO t1 VALUES (100, 300, 1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (150, 550, 1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (250, 150, 1);
INSERT 0 1
gaussdb=# ANALYZE t1;
ANALYZE
gaussdb=# ANALYZE t1((a,b));
ANALYZE
-- View the column statistics before import.
gaussdb=# SELECT staattnum,stadistinct,stakind1,stanumbers1,stavalues1 FROM PG_STATISTIC
WHERE starelid='t1'::REGCLASS;
 staattnum | stadistinct | stakind1 |        stanumbers1         |  stavalues1
-----------+-------------+----------+----------------------------+---------------
         1 |          -.5 |        1 | {.333333,.333333,.333333} | {100,150,250}
         2 |          -.5 |        1 | {.333333,.333333,.333333} | {150,300,550}
         3 |     -.166667 |        1 | {1}                       | {1}
(3 rows)
-- View expression statistics before import.
gaussdb=# SELECT staattnum,stadistinct,stakind1,stanumbers1,stavalues1 FROM PG_STATISTIC
WHERE starelid='idx_t1_expr'::REGCLASS;
 staattnum | stadistinct | stakind1 |        stanumbers1         |     stavalues1
-----------+-------------+----------+----------------------------+---------------------
         1 |          -.5 |        1 | {.333333,.333333,.333333} | {10000,22500,62500}
         2 |          -.5 |        1 | {.333333,.333333,.333333} | {200,300,500}
         3 |          -.5 |        1 | {.333333,.333333,.333333} | {30000,37500,82500}
(3 rows)
-- View the multi-column statistics before import.
gaussdb=# SELECT stakey,staextname,stadistinct,stakind1,stanumbers1,stavalues1 FROM
PG_STATISTIC_EXT WHERE starelid='t1'::REGCLASS;
 stakey |     staextname      | stadistinct | stakind1 | stanumbers1 | stavalues1
--------+---------------------+-------------+----------+-------------+------------
 1 2    | extname_-1308656218 |          -.5 |        8 |             |
(1 row)

-- Import single-column and multi-column statistics and view the system catalog.
gaussdb=# CALL DBE_STATS.IMPORT_COLUMN_STATS(ownname => 'dbe_stats_import', tabname =>
't1', colname=> 'a', partname => null, stattab => 'export_table', statid => 't1_c', force => false);
 import_column_stats
---------------------
(1 row)
gaussdb=# CALL DBE_STATS.IMPORT_COLUMN_STATS(ownname => 'dbe_stats_import', tabname =>
'idx_t1_expr', colname=> 'expr', partname => null, stattab => 'export_table', statid => 't1_expr', force
=> false);
 import_column_stats
---------------------
```



```
(1 row)
gaussdb=# CALL DBE_STATS.IMPORT_COLUMN_STATS(ownname => 'dbe_stats_import', tabname =>
't1', colname=> 'extname_-1308656218', partname => null, stattab => 'export_table', statid => 't1_c',
force => false);
 import_column_stats
---------------------
(1 row)

-- View the column-level statistics after import.
gaussdb=# SELECT staattnum,stadistinct,stakind1,stanumbers1,stavalues1 FROM PG_STATISTIC
WHERE starelid='t1'::REGCLASS;
 staattnum | stadistinct | stakind1 |        stanumbers1         |  stavalues1
-----------+-------------+----------+----------------------------+---------------
         1 |          -1 |        2 |                            | {100,150,250}
         2 |        -.5 |        1 | {.333333,.333333,.333333} | {150,300,550}
         3 |    -.166667 |        1 | {1}                        | {1}
(3 rows)
-- View the expression-level statistics after import.
gaussdb=# SELECT staattnum,stadistinct,stakind1,stanumbers1,stavalues1 FROM PG_STATISTIC
WHERE starelid='idx_t1_expr'::REGCLASS;
 staattnum | stadistinct | stakind1 | stanumbers1 |     stavalues1
-----------+-------------+----------+-------------+---------------------
         1 |          -1 |        2 |             | {10000,22500,62500}
         2 |          -1 |        2 |             | {200,300,500}
         3 |          -1 |        2 |             | {30000,37500,82500}
(3 rows)
-- View the multi-column statistics after import.
gaussdb=# SELECT stakey,staextname,stadistinct,stakind1,stanumbers1,stavalues1 FROM
PG_STATISTIC_EXT WHERE starelid='t1'::REGCLASS;
 stakey |     staextname      | stadistinct | stakind1 | stanumbers1 | stavalues1
--------+---------------------+-------------+----------+-------------+------------
 1 2    | extname_-1308656218 |          -1 |        8 |             |
(1 row)
-- Delete a table or namespace.
gaussdb=# DROP TABLE t1;
DROP TABLE
gaussdb=# DROP TABLE export_table;
DROP TABLE
gaussdb=# DROP SCHEMA dbe_stats_import;
DROP SCHEMA
```

- DBE_STATS.IMPORT_SCHEMA_STATS

  Retrieves statistics for the specified schema from the user statistics table of the stattab identifier and writes it back to the system catalog. This API does not return any value.

  The prototype of the DBE_STATS.IMPORT_SCHEMA_STATS function is as follows:

```
DBE_STATS.IMPORT_SCHEMA_STATS (
   ownname       VARCHAR2,
   stattab       VARCHAR2,
   statid        VARCHAR2 DEFAULT NULL,
   statown       VARCHAR2 DEFAULT NULL,
   no_invalidate BOOLEAN  DEFAULT NULL,
   force         BOOLEAN DEFAULT FALSE,
   stat_category VARCHAR2 DEFAULT NULL);
```


Table 10-391 DBE_STATS.IMPORT_SCHEMA_STATS parameters

Parameter	Type	Whether NULL Is Allowed	Description
ownname	VARCHAR2	No	Name of the schema. Null indicates that the current schema is used by default.
stattab	VARCHAR2	No	Name of the user table that stores statistics.
statid	VARCHAR2	Yes	(Optional) Identifier that points to statistics in stattab.
statown	VARCHAR2	Yes	Not supported currently.
no_invalidate	BOOLEAN	Yes	Not supported currently.
force	BOOLEAN	Yes	Specifies whether to forcibly import data. The lock status is ignored. The default value is **FALSE**.
stat_category	VARCHAR2	Yes	Not supported currently.

NOTE

- To call this procedure to import statistics, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- To call this procedure to import statistics from a user statistics table to a system catalog, you must have the SELECT, INSERT, DELETE, and UPDATE permissions on the user statistics table.
- You need to import statistics about all tables on which the current user has permission in the specified schema. Tables on which the current user does not have permission are skipped and no error is reported.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- The method of importing schema statistics is the same as that of importing table statistics. This section describes only the API calling method. For details, see the DBE_STATS.IMPORT_TABLE_STATS
```



```
example.
gaussdb=# CALL DBE_STATS.IMPORT_SCHEMA_STATS(ownname => 'dbe_stats_import', stattab =>
'export_table', statid => 's1',force => false);
 import_schema_stats
---------------------
(1 row)
```

- DBE_STATS.SET_COLUMN_STATS

  Sets column-related statistics, including single-column, multi-column, and expression statistics. This API does not return any value.

  The prototype of the DBE_STATS.SET_COLUMN_STATS function is as follows:

```
DBE_STATS.SET_COLUMN_STATS (
    ownname         VARCHAR2,
    tabname         VARCHAR2,
    colname         VARCHAR2,
    partname        VARCHAR2    DEFAULT NULL,
    stattab         VARCHAR2    DEFAULT NULL,
    statid          VARCHAR2    DEFAULT NULL,
    distcnt         NUMBER      DEFAULT NULL,
    density         NUMBER      DEFAULT NULL,
    nullcnt         NUMBER      DEFAULT NULL,
    srec            VARCHAR2    DEFAULT NULL,
    avgclen         NUMBER      DEFAULT NULL,
    flags           NUMBER      DEFAULT NULL,
    statown         VARCHAR2    DEFAULT NULL,
    no_invalidate   BOOLEAN     DEFAULT NULL,
    force           BOOLEAN     DEFAULT FALSE
);
```

Table 10-392 DBE_STATS.SET_COLUMN_STATS parameters

Parameter	Type	Whether NULL Is Allowed	Description
ownname	VARCHAR2	No	Schema where the table is located. Null indicates that the current schema is used by default.
tabname	VARCHAR2	No	Table name.
colname	VARCHAR2	No	Column name.
partname	VARCHAR2	Yes	Name of the table partition whose statistics are to be set. If the table has been partitioned and **partname** is null, global column–level statistics are set.

Parameter	Type	Whether NULL Is Allowed	Description
stattab	VARCHAR2	Yes	This parameter is not supported currently.
statid	VARCHAR2	Yes	This parameter is not supported currently.
distcnt	NUMBER	Yes	Number of NDVs in the column.
density	NUMBER	Yes	Not supported currently.
nullcnt	NUMBER	Yes	Number of null values in the column.
srec	VARCHAR2	Yes	This parameter is not supported currently.
avgclen	NUMBER	Yes	Average length (in bytes) of the fields in the column.
flags	NUMBER	Yes	This parameter is not supported currently.
statown	VARCHAR2	Yes	This parameter is not supported currently.
no_invalidate	BOOLEAN	Yes	This parameter is not supported currently.
force	NUMBER	Yes	Behavior when the statistics are locked. If the value is **true**, the procedure sets the value even if the column-level statistics are locked. The default value is **FALSE**.

NOTE

- To call this procedure to set column-level statistics for a table, you must have the same permissions as the ANALYZE statement on the tables involved in the column. For details, see **ANALYZE | ANALYSE**.
- This API can be used to set expression statistics. Expression creation depends on indexes. Therefore, when this API is called to set expression statistics, **tabname** must be set to the index name.
- Currently, only NDV (distcnt), nullfrac (nullcnt) and width (avgclen) statistics of columns can be set.
- After the statistics of a single column or expression are deleted, this API can still be called to set the statistics of the corresponding single column or expression. In this case, new data is inserted into the **PG_STATISTIC** system catalog. After the multi-column statistics are deleted, this API cannot be used to reset the statistics.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- Prerequisites: Create a table and collect statistics.
gaussdb=# CREATE SCHEMA dbe_stats_set_schema;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA = dbe_stats_set_schema;
SET
gaussdb=# DROP TABLE IF EXISTS t1;
NOTICE:  table "t1" does not exist, skipping
DROP TABLE
gaussdb=# CREATE TABLE IF NOT EXISTS t1 (a int, b int, c int);
CREATE TABLE
gaussdb=# CREATE INDEX idx_t1_expr ON t1((a*a), (a+a), (a*b));
CREATE INDEX
gaussdb=# INSERT INTO t1 VALUES (100, 300, 1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (150, 550, 1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (250, 150, 1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (250, 150, 1);
INSERT 0 1
gaussdb=# ANALYZE t1;
ANALYZE
gaussdb=# ANALYZE t1((a, b));
ANALYZE
-- Set single-column statistics.
-- Query the original statistics.
gaussdb=# SELECT staattnum,stanullfrac,stawidth,stadistinct FROM PG_STATISTIC WHERE
starelid='t1'::REGCLASS;
 staattnum | stanullfrac | stawidth | stadistinct
-----------+-------------+----------+-------------
         1 |           0 |        4 |       -.75
         2 |           0 |        4 |       -.75
         3 |           0 |        4 |       -.25
(3 rows)
-- Set the column-level statistics and query again.
gaussdb=# CALL DBE_STATS.SET_COLUMN_STATS(ownname => 'dbe_stats_set_schema', tabname =>
't1', colname => 'a', distcnt => 6, nullcnt => 5, avgclen => 42);
 set_column_stats
------------------

(1 row)
gaussdb=# SELECT staattnum,stanullfrac,stawidth,stadistinct FROM PG_STATISTIC WHERE
starelid='t1'::REGCLASS;
 staattnum | stanullfrac | stawidth | stadistinct
-----------+-------------+----------+-------------
         1 |           1 |       42 |           6
         2 |           0 |        4 |       -.75
         3 |           0 |        4 |       -.25
(3 rows)
```



```
-- Set multi-column statistics.
-- Query the original statistics.
gaussdb=# SELECT stakey,staextname,stanullfrac,stawidth,stadistinct FROM PG_STATISTIC_EXT
WHERE starelid='t1'::REGCLASS;
 stakey |     staextname      | stanullfrac | stawidth | stadistinct
--------+---------------------+-------------+----------+-------------
 1 2    | extname_-1308656218 |           0 |        8 |        -.75
(1 row)
-- Set the statistics and query again.
gaussdb=# CALL DBE_STATS.SET_COLUMN_STATS(ownname => 'dbe_stats_set_schema', tabname =>
't1', colname => 'extname_-1308656218', distcnt => 5, nullcnt => 5, avgclen => 2);
 set_column_stats
------------------
(1 row)
gaussdb=# SELECT stakey,staextname,stanullfrac,stawidth,stadistinct FROM PG_STATISTIC_EXT
WHERE starelid='t1'::REGCLASS;
 stakey |     staextname      | stanullfrac | stawidth | stadistinct
--------+---------------------+-------------+----------+-------------
 1 2    | extname_-1308656218 |           1 |        2 |           5
(1 row)
-- Set expression statistics.
-- Query the original statistics.
gaussdb=# SELECT staattnum,stanullfrac,stawidth,stadistinct FROM PG_STATISTIC WHERE
starelid='idx_t1_expr'::REGCLASS;
 staattnum | stanullfrac | stawidth | stadistinct
-----------+-------------+----------+-------------
         1 |           0 |        4 |        -.75
         2 |           0 |        4 |        -.75
         3 |           0 |        4 |        -.75
(3 rows)
-- Query an expression column name.
gaussdb=# SELECT attnum, attname FROM PG_ATTRIBUTE WHERE attrelid='idx_t1_expr'::REGCLASS;
 attnum | attname
--------+---------
      1 | expr
      2 | expr1
      3 | expr2
(3 rows)
-- Set the statistics and query again.
gaussdb=# CALL DBE_STATS.SET_COLUMN_STATS(ownname => 'dbe_stats_set_schema', tabname =>
'idx_t1_expr', colname => 'expr', distcnt => 13, nullcnt => 5, avgclen => 22);
 set_column_stats
------------------
(1 row)
gaussdb=# SELECT staattnum,stanullfrac,stawidth,stadistinct FROM PG_STATISTIC WHERE
starelid='idx_t1_expr'::REGCLASS;
 staattnum | stanullfrac | stawidth | stadistinct
-----------+-------------+----------+-------------
         1 |           1 |       22 |          13
         2 |           0 |        4 |        -.75
         3 |           0 |        4 |        -.75
(3 rows)
-- Delete a table or namespace.
gaussdb=# DROP INDEX idx_t1_expr;
DROP INDEX
gaussdb=# DROP TABLE t1;
DROP TABLE
gaussdb=# DROP SCHEMA dbe_stats_set_schema cascade;
DROP SCHEMA
```

- DBE_STATS.SET_INDEX_STATS

  Sets index-related statistics. This API does not return any value.

  The prototype of the DBE_STATS.SET_INDEX_STATS function is:

```
DBE_STATS.SET_INDEX_STATS (
   ownname       VARCHAR2,
   indname       VARCHAR2,
   partname      VARCHAR2  DEFAULT NULL,
   stattab       VARCHAR2  DEFAULT NULL,
```



```
  statid        VARCHAR2  DEFAULT NULL,
  numrows       NUMBER    DEFAULT NULL,
  numlblks      NUMBER    DEFAULT NULL,
  relallvisible NUMBER    DEFAULT NULL,
  numdist       NUMBER    DEFAULT NULL,
  avglblk       NUMBER    DEFAULT NULL,
  avgdblk       NUMBER    DEFAULT NULL,
  clstfct       NUMBER    DEFAULT NULL,
  indlevel      NUMBER    DEFAULT NULL,
  flags         NUMBER    DEFAULT NULL,
  statown       VARCHAR2  DEFAULT NULL,
  no_invalidate BOOLEAN   DEFAULT NULL,
  guessq        NUMBER    DEFAULT NULL,
  cachedblk     NUMBER    DEFAULT NULL,
  cachehit      NUMBER    DEFAULT NULL,
  force         BOOLEAN   DEFAULT FALSE
);
```

Table 10-393 DBE_STATS.SET_INDEX_STATS parameters

Parameter	Type	Whether NULL Is Allowed	Description
ownname	VARCHAR2	No	Schema to which the index belongs. Null indicates that the current schema is used by default.
indname	VARCHAR2	No	Index name.
partname	VARCHAR2	Yes	Name of the index partition whose statistics are to be set. If the index has been partitioned and **partname** is null, global index–level statistics are set.
stattab	VARCHAR2	Yes	This parameter is not supported currently.
statid	VARCHAR2	Yes	This parameter is not supported currently.
numrows	NUMBER	Yes	Number of rows in an index.
numlblks	NUMBER	Yes	Number of pages in an index.

Parameter	Type	Whether NULL Is Allowed	Description
relallvisible	NUMBER	Yes	Number of pages marked as all visible in the index.
numdist	NUMBER	Yes	This parameter is not supported currently.
avglblk	NUMBER	Yes	This parameter is not supported currently.
avgdblk	NUMBER	Yes	This parameter is not supported currently.
clstfct	NUMBER	Yes	This parameter is not supported currently.
indlevel	NUMBER	Yes	This parameter is not supported currently.
flags	NUMBER	Yes	This parameter is not supported currently.
statown	VARCHAR2	Yes	This parameter is not supported currently.
no_invalidate	BOOLEAN	Yes	This parameter is not supported currently.
guessq	NUMBER	Yes	This parameter is not supported currently.
cachedblk	NUMBER	Yes	This parameter is not supported currently.
cachehit	NUMBER	Yes	This parameter is not supported currently.

Parameter	Type	Whether NULL Is Allowed	Description
force	BOOLEAN	Yes	Behavior when the statistics are locked. If the value is **TRUE**, the procedure sets the value even if the index-level statistics are locked. The default value is **FALSE**.

NOTE

- To call this procedure to set index statistics, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- Currently, only reltuples (numrows), relpages (numlblks) and relallvisible (relallvisible) statistics of indexes can be set.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- Prerequisites: Create a table and collect statistics.
gaussdb=# CREATE SCHEMA dbe_stats_set_schema;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA = dbe_stats_set_schema;
SET
gaussdb=# DROP TABLE IF EXISTS t1;
NOTICE:  table "t1" does not exist, skipping
DROP TABLE
gaussdb=# CREATE TABLE IF NOT EXISTS t1 (a int, b int, c int);
CREATE TABLE
gaussdb=# CREATE INDEX idx_t1_a on t1(a);
CREATE INDEX
gaussdb=# INSERT INTO t1 VALUES (100, 300, 1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (150, 550, 1);
INSERT 0 1
gaussdb=# INSERT INTO t1 VALUES (250, 150, 1);
INSERT 0 1
gaussdb=# ANALYZE t1;
ANALYZE
-- Set index statistics.
-- View the current index statistics.
gaussdb=# SELECT reltuples,relpages,relallvisible FROM PG_CLASS WHERE relname='idx_t1_a';
 reltuples | relpages | relallvisible
-----------+----------+---------------
         3 |        2 |             0
(1 row)
-- Set the statistics and query again.
gaussdb=# CALL DBE_STATS.SET_INDEX_STATS(indname => 'idx_t1_a', ownname =>
'dbe_stats_set_schema', numrows => 1, numlblks => 3, relallvisible => 6);
 set_index_stats
-----------------
(1 row)
```



```
gaussdb=# SELECT reltuples,relpages,relallvisible FROM PG_CLASS WHERE relname='idx_t1_a';
 reltuples | relpages | relallvisible
-----------+----------+---------------
         1 |        3 |             6
(1 row)
-- Delete a table or namespace.
gaussdb=# DROP INDEX idx_t1_a;
DROP INDEX
gaussdb=# DROP TABLE t1;
DROP TABLE
gaussdb=# DROP SCHEMA dbe_stats_set_schema CASCADE;
DROP SCHEMA
```

- DBE_STATS.SET_TABLE_STATS

  Sets table-level statistics. This API does not return any value.

  The prototype of the DBE_STATS.SET_TABLE_STATS function is as follows:

```
DBE_STATS.SET_TABLE_STATS (
   ownname          VARCHAR2,
   tabname          VARCHAR2,
   partname         VARCHAR2 DEFAULT NULL,
   stattab          VARCHAR2  DEFAULT NULL,
   statid           VARCHAR2  DEFAULT NULL,
   numrows          NUMBER    DEFAULT NULL,
   numblks          NUMBER    DEFAULT NULL,
   relallvisible    NUMBER    DEFAULT NULL,
   avgrlen          NUMBER    DEFAULT NULL,
   flags            NUMBER    DEFAULT NULL,
   statown          VARCHAR2  DEFAULT NULL,
   no_invalidate    BOOLEAN   DEFAULT NULL,
   cachedblk        NUMBER    DEFAULT NULL,
   cachehit         NUMBER    DEFAULT NULL,
   force            BOOLEAN   DEFAULT FALSE,
   im_imcu_count    NUMBER    DEFAULT NULL,
   im_block_count NUMBER    DEFAULT NULL,
   scanrate         NUMBER    DEFAULT NULL
);
```

Table 10-394 DBE_STATS.SET_TABLE_STATS parameters

Parameter	Parameter	Whether NULL Is Allowed	Description
ownname	VARCHAR2	No	Schema where the table is located. Null indicates that the current schema is used by default.
tabname	VARCHAR2	No	Table name.

Parameter	Parameter	Whether NULL Is Allowed	Description
partname	VARCHAR2	Yes	Name of the table partition whose statistics are to be set. If the table has been partitioned and **partname** is null, global table–level statistics are set.
stattab	VARCHAR2	Yes	This parameter is not supported currently.
statid	VARCHAR2	Yes	This parameter is not supported currently.
numrows	NUMBER	Yes	Number of rows in a table.
numblks	NUMBER	Yes	Number of pages in a table.
relallvisible	NUMBER	Yes	Number of pages marked as all visible in a table.
avgrlen	NUMBER	Yes	This parameter is not supported currently.
flags	NUMBER	Yes	This parameter is not supported currently.
statown	VARCHAR2	Yes	This parameter is not supported currently.
no_invalidate	BOOLEAN	Yes	This parameter is not supported currently.
cachedblk	NUMBER	Yes	This parameter is not supported currently.
cachehit	NUMBER	Yes	This parameter is not supported currently.

Parameter	Parameter	Whether NULL Is Allowed	Description
force	BOOLEAN	Yes	Behavior when the statistics are locked. If the value is **TRUE**, the procedure sets the value even if the table statistics are locked. The default value is **FALSE**.
im_imcu_count	NUMBER	Yes	This parameter is not supported currently.
im_block_count	NUMBER	Yes	This parameter is not supported currently.
scanrate	NUMBER	Yes	This parameter is not supported currently.

NOTE

- To call this procedure to set table statistics, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- Currently, only reltuples (numrows), relpages (numblks) and relallvisible (relallvisible) statistics of tables can be set.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- The procedure for setting index statistics is similar to that for setting index statistics. The following describes only the API calling method. For details, see the DBE_STATS.SET_INDEX_STATS API example.
-- Set table-level statistics.
gaussdb=# CALL DBE_STATS.SET_TABLE_STATS(tabname => 't1', ownname => 'dbe_stats_set_schema', numrows => 1, numblks => 3, relallvisible => 6);
 set_table_stats
-----------------

(1 row)
```

- DBE_STATS.DELETE_COLUMN_STATS

  Deletes column-related statistics, including single-column, multi-column, and expression statistics. This API does not return any value.

  The prototype of the DBE_STATS.DELETE_COLUMN_STATS function is as follows:

```
DBE_STATS.DELETE_COLUMN_STATS (
    ownname         VARCHAR2,
    tabname         VARCHAR2,
```



```
  colname        VARCHAR2,
  partname       VARCHAR2 DEFAULT NULL,
  stattab        VARCHAR2 DEFAULT NULL,
  statid         VARCHAR2 DEFAULT NULL,
  cascade_parts  BOOLEAN  DEFAULT TRUE,
  statown        VARCHAR2 DEFAULT NULL,
  no_invalidate  BOOLEAN  DEFAULT NULL,
  force          BOOLEAN  DEFAULT FALSE,
  col_stat_type  VARCHAR2 DEFAULT NULL
);
```

Table 10-395 DBE_STATS.DELETE_COLUMN_STATS parameters

Parameter	Type	Whether NULL Is Allowed	Description
ownname	VARCHAR2	No	Schema where the table is located. Null indicates that the current schema is used by default.
tabname	VARCHAR2	No	Table name.
colname	VARCHAR2	No	Column name.
partname	VARCHAR2	Yes	Name of the table partition whose statistics are to be deleted. If the table has been partitioned and **partname** is **null**, global column–level statistics are deleted. If **cascade_parts** is **true**, column-level statistics of all partitions are deleted.
stattab	VARCHAR2	Yes	This parameter is not supported currently.
statid	VARCHAR2	Yes	This parameter is not supported currently.

Parameter	Type	Whether NULL Is Allowed	Description
cascade_parts	BOOLEAN	Yes	Specifies whether to delete the statistics of all partitions under a column when deleting the column-level statistics. The default value is **true**.
statown	VARCHAR2	Yes	This parameter is not supported currently.
no_invalidate	BOOLEAN	Yes	This parameter is not supported currently.
force	BOOLEAN	Yes	Behavior when the statistics are locked. If the value is **true**, the procedure deletes the column statistics even if they are locked. The default value is **false**.
col_stat_type	VARCHAR2	Yes	This parameter is not supported currently.

NOTE

- To call this procedure to delete column statistics, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- This API can be used to delete expression statistics. Expression creation depends on indexes. Therefore, when this API is called to delete expression statistics, **tabname** must be set to the index name.
- If the table is a level-2 partitioned table and **partname** is set to the level-1 partition name, and **cascade_parts** is set to **true**, statistics on all level-2 partitions under the level-1 partition will be deleted.
- Before deleting the statistics of multiple columns, query the **staextname** column in **PG_STATISTIC_EXT** to obtain the aliases of multiple columns and transfer them as the input parameter **colname**.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- Prerequisites: Create a table and collect statistics.
gaussdb=# CREATE SCHEMA dbe_stats_delete_schema;
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA = dbe_stats_delete_schema;
SET
gaussdb=# DROP TABLE IF EXISTS t2;
NOTICE:  table "t2" does not exist, skipping
DROP TABLE
gaussdb=# CREATE TABLE t2(a int, b int, c int) PARTITION BY RANGE(a) SUBPARTITION BY RANGE(b)
 (
   PARTITION p1 VALUES LESS THAN(200)
   (SUBPARTITION s1 VALUES LESS THAN (500),
    SUBPARTITION s2 VALUES LESS THAN (MAXVALUE)
   ),
   PARTITION p2 VALUES LESS THAN(500)
   (SUBPARTITION s3 VALUES LESS THAN (500),
    SUBPARTITION s4 VALUES LESS THAN (MAXVALUE)
   ),
   PARTITION p3 VALUES LESS THAN(MAXVALUE)
   ( SUBPARTITION s5 VALUES LESS THAN (500),
     SUBPARTITION s6 VALUES LESS THAN (MAXVALUE)
   ));
CREATE TABLE
gaussdb=# INSERT INTO t2 VALUES (100, 300, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (100, 600, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (300, 300, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (300, 600, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (600, 400, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (600, 600, 1);
INSERT 0 1
gaussdb=# ANALYZE t2;
ANALYZE
gaussdb=# ANALYZE t2((a, b));
ANALYZE

-- Delete the single-column statistics of table t2 and do not cascade partitions. The partition statistics still exist.
-- View statistics about table-level columns.
gaussdb=# SELECT staattnum,stadistinct,stakind1,stanumbers1,stavalues1 FROM PG_STATISTIC
WHERE starelid='t2'::REGCLASS;
 staattnum | stadistinct | stakind1 |        stanumbers1         |  stavalues1
-----------+-------------+----------+----------------------------+---------------
         1 |         -.5 |        1 | {.333333,.333333,.333333} | {100,300,600}
```



```
        2 |         -.5 |        1 | {.5,.333333}              | {600,300}
        3 |    -.166667 |        1 | {1}                       | {1}
(3 rows)
-- View statistics about partition-level columns.
gaussdb=# SELECT part.relname,s.staattnum,s.stadistinct,s.stakind1,s.stanumbers1,s.stavalues1 FROM
PG_STATISTIC s,PG_PARTITION part WHERE part.parentid='t2'::REGCLASS and s.starelid=part.oid;
 relname | staattnum | stadistinct | stakind1 | stanumbers1 | stavalues1
---------+-----------+-------------+----------+-------------+------------
 p1      |         1 |          -1 |        1 | {1}         | {100}
 p1      |         2 |          -1 |        2 |             | {300,600}
 p1      |         3 |          -1 |        1 | {1}         | {1}
 p2      |         1 |          -1 |        1 | {1}         | {300}
 p2      |         2 |          -1 |        2 |             | {300,600}
 p2      |         3 |          -1 |        1 | {1}         | {1}
 p3      |         1 |          -1 |        1 | {1}         | {600}
 p3      |         2 |          -1 |        2 |             | {400,600}
 p3      |         3 |          -1 |        1 | {1}         | {1}
(9 rows)
-- Delete the table-level single-column statistics and do not cascade partitions. The table-level
statistics do not exist, and the partition-level column statistics still exist.
gaussdb=# CALL DBE_STATS.DELETE_COLUMN_STATS(ownname => 'dbe_stats_delete_schema',
tabname => 't2', colname => 'a', partname => null, cascade_parts => false);
 delete_column_stats
---------------------
(1 row)
gaussdb=# SELECT staattnum,stadistinct,stakind1,stanumbers1,stavalues1 FROM PG_STATISTIC
WHERE starelid='t2'::REGCLASS;
 staattnum | stadistinct | stakind1 | stanumbers1  | stavalues1
-----------+-------------+----------+--------------+------------
         2 |         -.5 |        1 | {.5,.333333} | {600,300}
         3 |    -.166667 |        1 | {1}          | {1}
(2 rows)
gaussdb=# SELECT part.relname,s.staattnum,s.stadistinct,s.stakind1,s.stanumbers1,s.stavalues1 FROM
PG_STATISTIC s,PG_PARTITION part WHERE part.parentid='t2'::REGCLASS and s.starelid=part.oid;
 relname | staattnum | stadistinct | stakind1 | stanumbers1 | stavalues1
---------+-----------+-------------+----------+-------------+------------
 p1      |         1 |          -1 |        1 | {1}         | {100}
 p1      |         2 |          -1 |        2 |             | {300,600}
 p1      |         3 |          -1 |        1 | {1}         | {1}
 p2      |         1 |          -1 |        1 | {1}         | {300}
 p2      |         2 |          -1 |        2 |             | {300,600}
 p2      |         3 |          -1 |        1 | {1}         | {1}
 p3      |         1 |          -1 |        1 | {1}         | {600}
 p3      |         2 |          -1 |        2 |             | {400,600}
 p3      |         3 |          -1 |        1 | {1}         | {1}
(9 rows)

-- Delete the multi-column statistics of table t2 and cascade partitions. Both the column-level
statistics of cascaded partitions and partitions are deleted.
-- Query statistics.
gaussdb=# SELECT stakey,staextname,stadistinct,stakind1,stanumbers1,stavalues1 FROM
PG_STATISTIC_EXT WHERE starelid='t2'::REGCLASS;
 stakey |      staextname      | stadistinct | stakind1 | stanumbers1 | stavalues1
--------+----------------------+-------------+----------+-------------+------------
 1 2    | extname_-1308656218 |          -1 |        8 |             |
(1 row)
gaussdb=# SELECT
part.relname,s.stakey,s.staextname,s.stadistinct,s.stakind1,s.stanumbers1,s.stavalues1 FROM
PG_STATISTIC_EXT s,PG_PARTITION part WHERE part.parentid='t2'::REGCLASS and s.starelid=part.oid;
 relname | stakey |      staextname      | stadistinct | stakind1 | stanumbers1 | stavalues1
---------+--------+----------------------+-------------+----------+-------------+------------
 p1      | 1 2    | extname_-1308656218 |          -1 |        8 |             |
 p2      | 1 2    | extname_-1308656218 |          -1 |        8 |             |
 p3      | 1 2    | extname_-1308656218 |          -1 |        8 |             |
(3 rows)
-- Delete the multi-column statistics and query the statistics.
gaussdb=# CALL DBE_STATS.DELETE_COLUMN_STATS(ownname => 'dbe_stats_delete_schema',
tabname => 't2', colname => 'extname_-1308656218', partname => null, cascade_parts => true);
 delete_column_stats
```



```
--------------------
(1 row)
gaussdb=# SELECT stakey,staextname,stadistinct,stakind1,stanumbers1,stavalues1 FROM
PG_STATISTIC_EXT WHERE starelid='t2'::REGCLASS;
 stakey | staextname | stadistinct | stakind1 | stanumbers1 | stavalues1
--------+------------+-------------+----------+-------------+------------
(0 rows)
gaussdb=# SELECT
part.relname,s.stakey,s.staextname,s.stadistinct,s.stakind1,s.stanumbers1,s.stavalues1 FROM
PG_STATISTIC_EXT s,PG_PARTITION part WHERE part.parentid='t2'::REGCLASS and s.starelid=part.oid;
 relname | stakey | staextname | stadistinct | stakind1 | stanumbers1 | stavalues1
---------+--------+------------+-------------+----------+-------------+------------
(0 rows)

-- Delete the column-level statistics of the p1 partition. The table-level, p2, and p3 column statistics
are not affected.
-- Query statistics.
gaussdb=# SELECT staattnum,stadistinct,stakind1,stanumbers1,stavalues1 FROM PG_STATISTIC
WHERE starelid='t2'::REGCLASS and staattnum=2;
 staattnum | stadistinct | stakind1 | stanumbers1  | stavalues1
-----------+-------------+----------+--------------+------------
         2 |         -.5 |        1 | {.5,.333333} | {600,300}
(1 row)
gaussdb=# SELECT part.relname,s.staattnum,s.stadistinct,s.stakind1,s.stanumbers1,s.stavalues1 FROM
PG_STATISTIC s,PG_PARTITION part WHERE part.parentid='t2'::REGCLASS and s.starelid=part.oid and
s.staattnum=2;
 relname | staattnum | stadistinct | stakind1 | stanumbers1 | stavalues1
---------+-----------+-------------+----------+-------------+------------
 p1      |         2 |          -1 |        2 |             | {300,600}
 p2      |         2 |          -1 |        2 |             | {300,600}
 p3      |         2 |          -1 |        2 |             | {400,600}
(3 rows)
-- Delete the column statistics of the p1 partition and query the statistics.
gaussdb=# CALL DBE_STATS.DELETE_COLUMN_STATS(ownname => 'dbe_stats_delete_schema',
tabname => 't2', colname => 'b', partname => 'p1', cascade_parts => false);
 delete_column_stats
---------------------
(1 row)
gaussdb=# SELECT staattnum,stadistinct,stakind1,stanumbers1,stavalues1 FROM PG_STATISTIC
WHERE starelid='t2'::REGCLASS and staattnum=2;
 staattnum | stadistinct | stakind1 | stanumbers1  | stavalues1
-----------+-------------+----------+--------------+------------
         2 |         -.5 |        1 | {.5,.333333} | {600,300}
(1 row)
gaussdb=# SELECT part.relname,s.staattnum,s.stadistinct,s.stakind1,s.stanumbers1,s.stavalues1 FROM
PG_STATISTIC s,PG_PARTITION part WHERE part.parentid='t2'::REGCLASS and s.starelid=part.oid and
s.staattnum=2;
 relname | staattnum | stadistinct | stakind1 | stanumbers1 | stavalues1
---------+-----------+-------------+----------+-------------+------------
 p2      |         2 |          -1 |        2 |             | {300,600}
 p3      |         2 |          -1 |        2 |             | {400,600}
(2 rows)
-- Delete a table or namespace.
gaussdb=# DROP TABLE t2;
DROP TABLE
gaussdb=# DROP SCHEMA dbe_stats_delete_schema CASCADE;
DROP SCHEMA
```

- DBE_STATS.DELETE_INDEX_STATS

  Deletes index-related statistics. This API does not return any value.

  The prototype of the DBE_STATS.DELETE_INDEX_STATS function is:

```
DBE_STATS.DELETE_INDEX_STATS (
   ownname          VARCHAR2,
   indname          VARCHAR2,
   partname         VARCHAR2 DEFAULT NULL,
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   cascade_parts    BOOLEAN  DEFAULT TRUE,
```



```
  statown          VARCHAR2 DEFAULT NULL,
  no_invalidate    BOOLEAN  DEFAULT NULL,
  stattype         VARCHAR2 DEFAULT NULL,
  force            BOOLEAN  DEFAULT FALSE,
  stat_category    VARCHAR2 DEFAULT NULL
)
```

Table 10-396 DBE_STATS.DELETE_INDEX_STATS parameters

Parameter	Type	Whether NULL Is Allowed	Description
ownname	VARCHAR2	No	Schema to which the index belongs. Null indicates that the current schema is used by default.
indname	VARCHAR2	No	Index name.
partname	VARCHAR2	Yes	Name of the index partition whose statistics are to be deleted. If the index has been partitioned and **partname** is **null**, global index–level statistics are deleted. If **cascade_parts** is **true**, index-level statistics of all partitions are deleted.
stattab	VARCHAR2	Yes	This parameter is not supported currently.
statid	VARCHAR2	Yes	This parameter is not supported currently.

Parameter	Type	Whether NULL Is Allowed	Description
cascade_parts	BOOLEAN	Yes	Specifies whether to delete the statistics of all partitions under an index when deleting the index-level statistics. The default value is **true**.
statown	VARCHAR2	Yes	This parameter is not supported currently.
no_invalidate	BOOLEAN	Yes	This parameter is not supported currently.
stattype	VARCHAR2	Yes	This parameter is not supported currently.
force	BOOLEAN	Yes	Behavior when the statistics are locked. If the value is **true**, the procedure deletes the index statistics even if they are locked. The default value is **false**.
stat_category	VARCHAR2	Yes	This parameter is not supported currently.

NOTE

- To call this procedure to delete index statistics, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- Deleting index statistics is equivalent to set **reltuples**, **relpages**, and **relallvisible** to **0**.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- Prerequisites: Create a table.
gaussdb=# CREATE SCHEMA dbe_stats_delete_schema;
```



```
CREATE SCHEMA
gaussdb=# SET CURRENT_SCHEMA = dbe_stats_delete_schema;
SET
gaussdb=# DROP TABLE IF EXISTS t2;
NOTICE:  table "t2" does not exist, skipping
DROP TABLE
gaussdb=# CREATE TABLE t2(a int, b int, c int) PARTITION BY RANGE(a) SUBPARTITION BY RANGE(b)
 (
   PARTITION p1 VALUES LESS THAN(200)
   (SUBPARTITION s1 VALUES LESS THAN (500),
    SUBPARTITION s2 VALUES LESS THAN (MAXVALUE)
   ),
   PARTITION p2 VALUES LESS THAN(500)
   (SUBPARTITION s3 VALUES LESS THAN (500),
    SUBPARTITION s4 VALUES LESS THAN (MAXVALUE)
   ),
   PARTITION p3 VALUES LESS THAN(MAXVALUE)
   ( SUBPARTITION s5 VALUES LESS THAN (500),
     SUBPARTITION s6 VALUES LESS THAN (MAXVALUE)
   ));
CREATE TABLE
gaussdb=# CREATE INDEX idx_t2_local_a ON t2(a) LOCAL
 (
    PARTITION p1_idx_a
    (
       SUBPARTITION s1_idx_a,
       SUBPARTITION s2_idx_a
    ),
    PARTITION p2_idx_a
    (
       SUBPARTITION s3_idx_a,
       SUBPARTITION s4_idx_a
    ),
    PARTITION p3_idx_a
    (
       SUBPARTITION s5_idx_a,
       SUBPARTITION s6_idx_a
    ));
CREATE INDEX
gaussdb=# INSERT INTO t2 VALUES (100, 300, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (100, 600, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (300, 300, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (300, 600, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (600, 400, 1);
INSERT 0 1
gaussdb=# INSERT INTO t2 VALUES (600, 600, 1);
INSERT 0 1
gaussdb=# ANALYZE t2;
ANALYZE
gaussdb=# ANALYZE t2((a, b));
ANALYZE

-- Delete the statistics of the idx_t2_local_a index without cascading the statistics of the partitioned
index.
-- View statistics.
gaussdb=# SELECT relpages,reltuples FROM PG_CLASS WHERE relname='idx_t2_local_a';
 relpages | reltuples
----------+-----------
        6 |         6
(1 row)
gaussdb=# SELECT relname,relpages,reltuples FROM PG_PARTITION WHERE
parentid='idx_t2_local_a'::REGCLASS;
 relname  | relpages | reltuples
----------+----------+-----------
 s1_idx_a |        1 |         0
```



```
 s2_idx_a |        1 |         0
 s3_idx_a |        1 |         0
 s4_idx_a |        1 |         0
 s5_idx_a |        1 |         0
 s6_idx_a |        1 |         0
(6 rows)
-- Delete statistics and query the system catalog.
gaussdb=# CALL DBE_STATS.DELETE_INDEX_STATS(ownname => 'dbe_stats_delete_schema', indname
=> 'idx_t2_local_a', partname => null, cascade_parts => false);
 delete_index_stats
--------------------

(1 row)
gaussdb=# SELECT relpages,reltuples FROM PG_CLASS WHERE relname='idx_t2_local_a';
 relpages | reltuples
----------+-----------
        0 |         0
(1 row)
gaussdb=# SELECT relname,relpages,reltuples FROM PG_PARTITION WHERE
parentid='idx_t2_local_a'::REGCLASS;
 relname  | relpages | reltuples
----------+----------+-----------
 s1_idx_a |        1 |         0
 s2_idx_a |        1 |         0
 s3_idx_a |        1 |         0
 s4_idx_a |        1 |         0
 s5_idx_a |        1 |         0
 s6_idx_a |        1 |         0
(6 rows)

-- Delete statistics about the s1_idx_a partition of the idx_t2_local_a index.
gaussdb=# CALL DBE_STATS.DELETE_INDEX_STATS(ownname => 'dbe_stats_delete_schema', indname
=> 'idx_t2_local_a', partname => 's1_idx_a', cascade_parts => false);
 delete_index_stats
--------------------

(1 row)
gaussdb=# SELECT relname,relpages,reltuples FROM PG_PARTITION WHERE
parentid='idx_t2_local_a'::REGCLASS;
 relname  | relpages | reltuples
----------+----------+-----------
 s2_idx_a |        1 |         0
 s3_idx_a |        1 |         0
 s4_idx_a |        1 |         0
 s5_idx_a |        1 |         0
 s6_idx_a |        1 |         0
 s1_idx_a |        0 |         0
(6 rows)

-- Delete index statistics of all partitions in cascading mode.
gaussdb=# CALL DBE_STATS.DELETE_INDEX_STATS(ownname => 'dbe_stats_delete_schema', indname
=> 'idx_t2_local_a', partname => null, cascade_parts => true);
 delete_index_stats
--------------------

(1 row)
gaussdb=# SELECT relname,relpages,reltuples FROM PG_PARTITION WHERE
parentid='idx_t2_local_a'::REGCLASS;
 relname  | relpages | reltuples
----------+----------+-----------
 s6_idx_a |        0 |         0
 s5_idx_a |        0 |         0
 s4_idx_a |        0 |         0
 s3_idx_a |        0 |         0
 s2_idx_a |        0 |         0
 s1_idx_a |        0 |         0
(6 rows)
-- Delete a table or namespace.
gaussdb=# DROP INDEX idx_t2_local_a;
DROP INDEX
gaussdb=# DROP TABLE t2;
DROP TABLE
```



```
gaussdb=# DROP SCHEMA dbe_stats_delete_schema CASCADE;
DROP SCHEMA
```

- DBE_STATS.DELETE_TABLE_STATS

  Deletes table-level statistics. This API does not return any value.

  The prototype of the DBE_STATS.DELETE_TABLE_STATS function is:

```
DBE_STATS.DELETE_TABLE_STATS (
   ownname              VARCHAR2,
   tabname              VARCHAR2,
   partname             VARCHAR2 DEFAULT NULL,
   stattab              VARCHAR2 DEFAULT NULL,
   statid               VARCHAR2 DEFAULT NULL,
   cascade_parts        BOOLEAN  DEFAULT TRUE,
   cascade_columns      BOOLEAN  DEFAULT TRUE,
   cascade_indexes      BOOLEAN  DEFAULT TRUE,
   statown              VARCHAR2 DEFAULT NULL,
   no_invalidate        BOOLEAN  DEFAULT NULL,
   force                BOOLEAN  DEFAULT FALSE,
   stat_category        VARCHAR2 DEFAULT NULL
)
```

Table 10-397 DBE_STATS.DELETE_TABLE_STATS parameters

Parameter	Type	Whether NULL Is Allowed	Description
ownname	VARCHAR2	No	Schema where the table is located. Null indicates that the current schema is used by default.
tabname	VARCHAR2	No	Table name.
partname	VARCHAR2	Yes	Name of the table partition whose statistics are to be deleted. If the table has been partitioned and **partname** is **null**, global table–level statistics are deleted. If **cascade_parts** is **true**, table-level statistics of all partitions are deleted.
stattab	VARCHAR2	Yes	This parameter is not supported currently.

Parameter	Type	Whether NULL Is Allowed	Description
statid	VARCHAR2	Yes	This parameter is not supported currently.
cascade_parts	BOOLEAN	Yes	Specifies whether to delete the statistics of all table partitions when deleting the table-level statistics. The default value is **true**.
cascade_columns	BOOLEAN	Yes	Specifies whether to call delete_column_stats to delete all column-level statistics of the table. The default value is **true**.
cascade_indexes	BOOLEAN	Yes	Specifies whether to call delete_index_stats to delete all index-level statistics of the table. The default value is **true**.
statown	VARCHAR2	Yes	This parameter is not supported currently.
no_invalidate	BOOLEAN	Yes	This parameter is not supported currently.

Parameter	Type	Whether NULL Is Allowed	Description
force	BOOLEAN	Yes	Behavior when the statistics are locked. If the value is **true**, the procedure deletes the table statistics even if they are locked. The default value is **false**.
stat_category	VARCHAR2	Yes	This parameter is not supported currently.

NOTE

- To call this procedure to delete table statistics, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- If **cascade_columns** or **cascade_indexes** is set to **true**, **cascade_parts** also determines whether to delete the statistics of the corresponding partition key or index in cascading mode. For example, if **cascade_parts** is set to **true**, not only table-level statistics of all partitions of a partitioned table are deleted, but also column-level and index-level statistics of all partitions are deleted.
- Deleting table statistics is equivalent to clearing the information of reltuples, relpages, and relallvisible.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- The procedure for deleting table-level index statistics is similar to that for deleting index statistics. The following describes only the API calling method. For details, see the DBE_STATS.DELETE_INDEX_STATS example.
-- Delete table-level statistics, including all column and index statistics of a table.
gaussdb=# CALL DBE_STATS.DELETE_TABLE_STATS(ownname => 'dbe_stats_delete_schema', tabname => 't2', cascade_columns => true, cascade_indexes => true);
 delete_table_stats
--------------------
(1 row)
```

- DBE_STATS.DELETE_SCHEMA_STATS

  Deletes all table-level, index-level, and column-level statistics in a schema. This API does not return any value.

  The prototype of the DBE_STATS.DELETE_SCHEMA_STATS function is:

```
DBE_STATS.DELETE_SCHEMA_STATS (
   ownname          VARCHAR2,
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   statown          VARCHAR2 DEFAULT NULL,
   no_invalidate    BOOLEAN  DEFAULT NULL,
   force            BOOLEAN  DEFAULT FALSE,
   stat_category    VARCHAR2 DEFAULT NULL
);
```


Table 10-398 DBE_STATS.DELETE_SCHEMA_STATS parameters

Parameter	Type	Whether NULL Is Allowed	Description
ownname	VARCHAR2	No	Schema name. Null indicates that the current schema is used by default.
stattab	VARCHAR2	Yes	This parameter is not supported currently.
statid	VARCHAR2	Yes	This parameter is not supported currently.
statown	VARCHAR2	Yes	This parameter is not supported currently.
no_invalidate	BOOLEAN	Yes	This parameter is not supported currently.
force	BOOLEAN	Yes	Behavior when the statistics are locked. If the value is **true**, the procedure deletes the statistics even if they are locked. The default value is **false**.
stat_category	VARCHAR2	Yes	This parameter is not supported currently.

NOTE

- To call this procedure to delete statistics, you must have the same permission on the specified table as that of the ANALYZE statement. For details, see **ANALYZE | ANALYSE**.
- If you delete statistics in a schema, that is, delete all tables on which the current user has permission in the specified schema, tables on which the current user does not have permission are skipped and no error is reported.
- If **ownname** is set to **null**, the current schema is used. In this case, you need to set **behavior_compat_options** to **bind_procedure_searchpath**.

Example:

```
-- Only API calling is displayed here. For details about the implementation cases, see
DBE_STATS.DELETE_TABLE_STATS.
```



```
gaussdb=# CALL DBE_STATS.DELETE_SCHEMA_STATS(ownname => 'dbe_stats_delete_schema');
 delete_schema_stats
---------------------
(1 row)
```

10.12.2.17 DBE_TASK

API Description

Table 10-399 lists all APIs supported by the **DBE_TASK** package.

Table 10-399 DBE_TASK

API	Description
DBE_TASK.SUBMIT	Submits a scheduled task. The job ID is automatically generated by the system.
DBE_TASK.JOB_SUBMIT	Same as DBE_TASK.SUBMIT. However, It provides syntax compatibility parameters.
DBE_TASK.ID_SUBMIT	Submits a scheduled task. The job ID is specified by the user.
DBE_TASK.CANCEL	Removes a scheduled task by job ID.
DBE_TASK.RUN	Executes a scheduled task.
DBE_TASK.FINISH	Disables or enables scheduled task execution.
DBE_TASK.UPDATE	Modifies user-definable attributes of a scheduled task, including the task content, next-execution time, and execution interval.
DBE_TASK.CHANGE	Same as DBE_TASK.UPDATE. However, It provides syntax compatibility parameters.
DBE_TASK.CONTENT	Modifies the content attribute of a scheduled task.
DBE_TASK.NEXT_TIME	Modifies the next-execution time attribute of a scheduled task.
DBE_TASK.INTERVAL	Modifies the execution interval attribute of a scheduled task.

- DBE_TASK.SUBMIT

  The stored procedure SUBMIT submits a scheduled task provided by the system.

  The prototype of the DBE_TASK.SUBMIT function is as follows:

  ```
  DBE_TASK.SUBMIT(
  what          IN   TEXT,
  next_time     IN   TIMESTAMP DEFAULT sysdate,
  interval_time IN   TEXT  DEFAULT 'null',
  id            OUT  INTEGER
  )RETURN INTEGER;
  ```


NOTE

When a scheduled task is created (using **DBE_TASK**), the system binds the current database and the username to the task by default. The API function can be called using CALL or SELECT. If the API function is called using CALLA, you do not need to set the output parameter. If the API function is called using SELECT, you need to set the output parameter. To call this function within a stored procedure, use **perform**. If the committed SQL statement task uses a non-public schema, specify the schema to a table schema or a function schema, or add **set current_schema = xxx** before the SQL statement.

Table 10-400 DBE_TASK.SUBMIT parameters

Parameter	Type	Input/Output Parameter	Whether NULL Is Allowed	Description
what	text	IN	No	SQL statement to be executed. One or multiple DDLs (excluding database-related operations), DMLs, anonymous blocks, and statements for calling stored procedures, or all four combined are supported.
next_time	timestamp	IN	No	Specifies the next time the job will be executed. The default value is the current system time (**sysdate**). If the specified time has past, the job is executed at the time it is submitted.
interval_time	text	IN	Yes	Calculates the next time to execute the job. It can be an interval expression, or **sysdate** followed by a numeric value, for example, **sysdate+1.0/24**. If this parameter is left empty or set to **null**, the job will be executed only once, and the job status will change to **'d'** afterward.
id	integer	OUT	No	Specifies the job ID. The value ranges from 1 to 32767. When SELECT is used for calling, this parameter cannot be added. When CALL is used for calling, this parameter must be added.

NOTICE

When you create a user using the **what** parameter, the plaintext password of the user is recorded in the log. Therefore, you are advised not to do so.

Example:

```
gaussdb=# SELECT DBE_TASK.SUBMIT('call pro_xxx();', to_date('20180101','yyyymmdd'),'sysdate+1');
 submit
--------
  31031
(1 row)

gaussdb=# SELECT DBE_TASK.SUBMIT('call pro_xxx();', to_date('20180101','yyyymmdd'),'sysdate
+1.0/24');
 submit
--------
    512
(1 row)

gaussdb=# DECLARE
gaussdb-#     jobid int;
gaussdb-# BEGIN
gaussdb$#     PERFORM DBE_TASK.SUBMIT('call pro_xxx();', sysdate, 'interval ''5 minute''', jobid);
gaussdb$# END;
gaussdb$# /
ANONYMOUS BLOCK EXECUTE
```

- DBE_TASK.JOB_SUBMIT

  The stored procedure SUBMIT submits a scheduled task provided by the system. In addition, it provides additional compatibility parameters.

  The prototype of the DBE_TASK.JOB_SUBMIT function is as follows:

```
DBE_TASK.JOB_SUBMIT(
job           OUT  INTEGER,
what           IN   TEXT,
next_date      IN   TIMESTAMP  DEFAULT sysdate,
job_interval  IN   TEXT       DEFAULT 'null',
no_parse       IN   BOOLEAN    DEFAULT false,
instance       IN   INTEGER    DEFAULT 0,
force          IN   BOOLEAN    DEFAULT false
)RETURN INTEGER;
```

Table 10-401 DBE_TASK.JOB_SUBMIT parameters

Parameter	Type	Input/Output Parameter	Whether NULL Is Allowed	Description
job	integer	OUT	No	Specifies the job ID. The value ranges from 1 to 32767. When dbe.job_submit is called by using the SELECT statement, this parameter can be omitted.
what	text	IN	No	SQL statement to be executed. One or multiple DDLs (excluding database-related operations), DMLs, anonymous blocks, and statements for calling stored procedures, or all four combined are supported.

Parameter	Type	Input/Output Parameter	Whether NULL Is Allowed	Description
next_date	timestamp	IN	Yes	Specifies the next time the job will be executed. The default value is the current system time (**sysdate**). If the specified time has past, the job is executed at the time it is submitted.
job_interval	text	IN	Yes	Calculates the next time to execute the job. It can be an interval expression, or **sysdate** followed by a numeric value, for example, **sysdate+1.0/24**. If this parameter is left empty or set to **null**, the job will be executed only once, and the job status will change to **'d'** afterward.
no_parse	Boolean	IN	Yes	The default value is **FALSE**, which is used only for syntax compatibility.
instance	integer	IN	Yes	The default value is **0**, which is used only for syntax compatibility.
force	Boolean	IN	Yes	The default value is **FALSE**, which is used only for syntax compatibility.

Example:

```
gaussdb=# DECLARE
gaussdb-#     id integer;
gaussdb-# BEGIN
gaussdb$#     id = DBE_TASK.JOB_SUBMIT(
gaussdb$#         what => 'insert into t1 values (1, 2)',
gaussdb$#         job_interval => 'sysdate + 1' --daily
gaussdb$#     );
gaussdb$# END;
gaussdb$# /
ANONYMOUS BLOCK EXECUTE
```

- DBE_TASK.ID_SUBMIT

  ID_SUBMIT has the same syntax function as SUBMIT, but the first parameter of ID_SUBMIT is an input parameter, that is, a specified job ID. In contrast, that last parameter of ID_SUBMIT is an output parameter, indicating the job ID automatically generated by the system.

  ```
  DBE_TASK.ID_SUBMIT(
  id            IN   BIGINT,
  what          IN   TEXT,
  next_time     IN   TIMESTAMP DEFAULT sysdate,
  interval_time IN   TEXT  DEFAULT 'null');
  ```

  Example:

  ```
  gaussdb=# CALL dbe_task.id_submit(101, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');
   id_submit
  ```



```
-----------

(1 row)
```

- DBE_TASK.CANCEL

  The stored procedure CANCEL deletes a specified task.

  The prototype of the DBE_TASK.CANCEL function is as follows:

  ```
  CANCEL(id  IN  INTEGER);
  ```

  Table 10-402 DBE_TASK.CANCEL parameters

Parameter	Type	Input/Output Parameter	Whether NULL Is Allowed	Description
id	integer	IN	No	Specifies the job ID.

  Example:

  ```
  gaussdb=# CALL dbe_task.cancel(101);
   cancel
  --------

  (1 row)
  ```

- DBE_TASK.RUN

  The stored procedure runs a scheduled task.

  The prototype of the DBE_TASK.RUN function is as follows:

  ```
  DBE_TASK.RUN(
  job          IN   BIGINT,
  force        IN   BOOLEAN DEFAULT FALSE);
  ```

  Table 10-403 DBE_TASK.RUN parameters

Parameter	Type	Input/Output Parameter	Whether NULL Is Allowed	Description
job	bigint	IN	No	Specifies the job ID.
force	Boolean	IN	Yes	Used only for syntax compatibility.

  Example:

  ```
  gaussdb=# BEGIN
  gaussdb$#     DBE_TASK.ID_SUBMIT(12345, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');
  ```



```
gaussdb$#     DBE_TASK.RUN(12345);
gaussdb$# END;
gaussdb$# /
ANONYMOUS BLOCK EXECUTE
```

- DBE_TASK.FINISH

  The stored procedure FINISH disables or enables a scheduled task.

  The prototype of the DBE_TASK.FINISH function is as follows:

```
DBE_TASK.FINISH(
id          IN   INTEGER,
broken      IN   BOOLEAN,
next_time   IN   TIMESTAMP  DEFAULT  sysdate);
```

Table 10-404 DBE_TASK.FINISH parameters

Parameter	Type	Input/Output Parameter	Whether NULL Is Allowed	Description
id	integer	IN	No	Specifies the job ID.
broken	Boolean	IN	No	Specifies the status flag, **true** for broken and **false** for not broken. The current job is updated based on the parameter value **true** or **false**. If the parameter is left empty, the job status remains unchanged.
next_time	timestamp	IN	Yes	Specifies the next execution time. The default value is the current system time. If **broken** is set to **true**, **next_time** is updated to **'4000-1-1'**. If **broken** is set to **false** and **next_time** is not empty, **next_time** is updated for the job. If **next_time** is empty, it will not be updated. This parameter can be omitted, and its default value will be used in this case.

Example:

```
gaussdb=# CALL dbe_task.id_submit(101, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');
 id_submit
-----------


(1 row)

gaussdb=# CALL dbe_task.finish(101, true);
 finish
--------
```



```
(1 row)
```

- DBE_TASK.UPDATE

  The stored procedure UPDATE modifies user-definable attributes of a task, including the task content, next-execution time, and execution interval.

  The prototype of the DBE_TASK.UPDATE function is as follows:

```
DBE_TASK.UPDATE(
id              IN     INTEGER,
content         IN     TEXT,
next_time       IN     TIMESTAMP,
interval_time   IN  TEXT);
```

Table 10-405 DBE_TASK.UPDATE parameters

Parameter	Type	Input/ Output Parameter	Whether NULL Is Allowed	Description
id	integer	IN	No	Specifies the job ID.
content	text	IN	Yes	Specifies the name of the stored procedure or SQL statement block that is executed. If this parameter is left empty, the system does not update the **content** parameter for the specified job. Otherwise, the system updates the **content** parameter for the specified job.
next_time	timestamp	IN	Yes	Specifies the next execution time. If this parameter is left empty, the system does not update the **next_time** parameter for the specified job. Otherwise, the system updates the **next_time** parameter for the specified job.
interval_time	text	IN	Yes	Specifies the time expression for calculating the next time the job will be executed. If this parameter is left empty, the system does not update the **interval_time** parameter for the specified job. Otherwise, the system updates the **interval_time** parameter for the specified job after necessary validity check. If this parameter is set to **"null"**, the job will be executed only once, and the job state will change to **'d'** afterward.

Example:


```
gaussdb=# CALL dbe_task.update(101, 'call userproc();', sysdate, 'sysdate + 1.0/1440')
 update
--------

(1 row)

gaussdb=# CALL dbe_task.update(101, 'insert into tbl_a values(sysdate);', sysdate, 'sysdate +
1.0/1440');
 update
--------

(1 row)
```

- DBE_TASK.CHANGE

  The stored procedure UPDATE modifies user-definable attributes of a task, including the task content, next-execution time, and execution interval.

  The prototype of the DBE_TASK.CHANGE function is as follows:

```
DBE_TASK.CHANGE(
job             IN    INTEGER,
what            IN    TEXT       DEFAULT NULL,
next_date       IN    TIMESTAMP  DEFAULT NULL,
job_interval    IN    TEXT       DEFAULT NULL,
instance        IN    INTEGER    DEFAULT NULL,
force           IN    BOOLEAN    DEFAULT false);
```

Table 10-406 DBE_TASK.CHANGE parameters

Parameter	Type	Input/ Output Parameter	Whether NULL Is Allowed	Description
job	integer	IN	No	Specifies the job ID.
what	text	IN	Yes	Specifies the name of the stored procedure or SQL statement block that is executed. If this parameter is left empty, the system does not update the **what** parameter for the specified job. Otherwise, the system updates the **what** parameter for the specified job.
next_date	timestamp	IN	Yes	Specifies the next execution time. If this parameter is left empty, the system does not update the **next_time** parameter for the specified job. Otherwise, the system updates the **next_date** parameter for the specified job.

Parameter	Type	Input/ Output Parameter	Whether NULL Is Allowed	Description
job_interval	text	IN	Yes	Specifies the time expression for calculating the next time the job will be executed. If this parameter is left empty, the system does not update the **job_interval** parameter for the specified job. Otherwise, the system updates the **job_interval** parameter for the specified job after necessary validity check. If this parameter is set to **"null"**, the job will be executed only once, and the job state will change to **'d'** afterward.
instance	integer	IN	Yes	Used only for syntax compatibility.
force	Boolean	IN	No	Used only for syntax compatibility.

Example:

```
gaussdb=# BEGIN
gaussdb$#     DBE_TASK.CHANGE(
gaussdb$#         job => 101,
gaussdb$#         what => 'insert into t2 values (2);'
gaussdb$#     );
gaussdb$# END;
gaussdb$# /
ANONYMOUS BLOCK EXECUTE
```

- DBE_TASK.CONTENT

  The stored procedure CONTENT modifies the procedures to be executed by a specified task.

  The prototype of the DBE_TASK.CONTENT function is as follows:

```
DBE_TASK.CONTENT(
id          IN     INTEGER,
content      IN TEXT);
```

Table 10-407 DBE_TASK.CONTENT parameters

Parameter	Type	Input/ Output Parameter	Whether NULL Is Allowed	Description
id	integer	IN	No	Specifies the job ID.

Parameter	Type	Input/ Output Parameter	Whether NULL Is Allowed	Description
content	text	IN	No	Specifies the name of the stored procedure or SQL statement block that is executed.

NOTE

- If the value specified by the **content** parameter is one or multiple executable SQL statements, program blocks, or stored procedures, this procedure can be executed successfully; otherwise, it will fail to be executed.
- If the value specified by the **content** parameter is a simple statement such as INSERT and UPDATE, a schema name must be added in front of the table name.

Example:

```
gaussdb=# CALL dbe_task.content(101, 'call userproc();');
 content
---------

(1 row)

gaussdb=# CALL dbe_task.content(101, 'insert into tbl_a values(sysdate);');
 content
---------

(1 row)
```

- DBE_TASK.NEXT_TIME

  The stored procedure NEXT_TIME modifies the next-execution time attribute of a task.

  The prototype of the DBE_TASK.NEXT_TIME function is as follows:

```
DBE_TASK.NEXT_TIME(
id         IN    BIGINT,
next_time  IN    TEXT);
```

Table 10-408 DBE_TASK.NEXT_TIME parameters

Parameter	Type	Input/ Output Parameter	Whether NULL Is Allowed	Description
id	bigint	IN	No	Specifies the job ID.
next_time	text	IN	No	Specifies the next execution time.

NOTE

If the specified **next_time** value is earlier than the current date, the job is executed once immediately.

Example:

```
gaussdb=# CALL dbe_task.next_time(101, sysdate);
 next_time
-----------

(1 row)
```

- DBE_TASK.INTERVAL

  The stored procedure INTERVAL modifies the execution interval attribute of a task.

  The prototype of the DBE_TASK.INTERVAL function is as follows:

```
DBE_TASK.INTERVAL(
id             IN      INTEGER,
interval_time      IN    TEXT);
```

Table 10-409 DBE_TASK.INTERVAL parameters

Parameter	Type	Input / Output Parameter	Whether NULL Is Allowed	Description
id	integer	IN	No	Specifies the job ID.
interval_time	text	IN	Yes	Specifies the time expression for calculating the next time the job will be executed. If this parameter is left empty or set to **null**, the job will be executed only once, and the job status will change to **'d'** afterward. The job interval must be of a valid time type or interval type.

Example:

```
gaussdb=# CALL dbe_task.interval(101, 'sysdate + 1.0/1440');
 interval
----------

(1 row)

gaussdb=# CALL dbe_task.cancel(101);
 cancel
--------

(1 row)
```

NOTE

For a job that is currently running (that is, **job_status** is **'r'**), it is not allowed to use the cancel, update, next_time, content, or interval API to delete or modify job parameters.

Constraints

1. You can create, update, and delete jobs only using the procedures provided by the **dbe_task** package. These procedures synchronize job information between different primary database nodes and associate primary keys between the **pg_job** and **pg_job_proc** catalogs. If you use DML statements to add, delete, or modify records in the **pg_job** catalog, job information will become inconsistent between primary database nodes and system catalogs may fail to be associated, compromising internal job management.
2. Each task created by a user is bound to a primary database node. If the primary database node fails while a task is being executed, the task status cannot be updated in real time and will stay at **'r'**. The task status will be updated to **'s'** only after the primary database node recovers. When a primary database node fails, all tasks on this primary database node cannot be scheduled or executed until the primary database node is restored manually, or deleted and then replaced.
3. For each job, the bound primary database node updates the real-time job information (including the job status, last execution start time, last execution end time, next execution start time, the number of execution failures [if any]) to the pg_job system catalog, and synchronizes the information to other primary database nodes, ensuring consistent job information between different primary database nodes. In the case of primary database node failures, job information synchronization is reattempted by the hosting primary database nodes, which increases job execution time. Although job information fails to be synchronized between primary database nodes, job information can still be properly updated in the pg_job table on the hosting primary database nodes, and jobs can be executed successfully. After a primary database node recovers, job information such as job execution time and status in its pg_job table may be incorrect and will be updated only after the jobs are executed again on related primary database nodes.
4. For each job, a thread is established to execute it. If multiple jobs are triggered concurrently as scheduled, the system will need some time to start the required threads, resulting in a latency of 0.1 ms in job execution.

10.12.2.18 DBE_UTILITY

API Description

Table 10-410 provides all APIs supported by the **DBE_UTILITY** package.

Table 10-410 DBE_UTILITY

API	Description
DBE_UTILITY.FORMAT_ERROR_BACKTRACE	Outputs the call stack of an abnormal stored procedure.
DBE_UTILITY.FORMAT_ERROR_STACK	Outputs detailed information about a stored procedure exception.
DBE_UTILITY.FORMAT_CALL_STACK	Outputs the call stack of a stored procedure.

API	Description
DBE_UTILITY.GET_TIME	Outputs the current time, which is used to obtain the execution duration.
DBE_UTILITY.CANONICALIZE	Canonicalizes the character string of a table name.
DBE_UTILITY.COMMA_TO_TABLE	Converts a comma-delimited string of names into a PL/SQL table of names.
DBE_UTILITY.DB_VERSION	Returns the version number and compatibility version number of the database.
DBE_UTILITY.EXEC_DDL_STATEMENT	Executes DDL statements entered by users.
DBE_UTILITY.EXPAND_SQL_TEXT_PROC	Expands the view of the SQL query.
DBE_UTILITY.GET_CPU_TIME	Returns the measured value of the current CPU processing time.
DBE_UTILITY.GET_ENDIANNESS	Obtains the big-endian and little-endian information of the byte order on the platform where the database is located.
DBE_UTILITY.GET_HASH_VALUE	Returns the hash value of a given string.
DBE_UTILITY.GET_SQL_HASH	Outputs the hash value of a given string. This stored procedure is used when **proc_outparam_override** is not enabled.
DBE_UTILITY.IS_BIT_SET	Checks whether parameter **n** exists in **r**.
DBE_UTILITY.IS_CLUSTER_DATABASE	Determines whether the current database is running in database cluster mode.
DBE_UTILITY.NAME_RESOLVE	Parses the given object name, including synonym translation and necessary authorization checks.
DBE_UTILITY.NAME_TOKENIZE	Parses the name in the **a [. b [. c]][@ dblink]** format.
DBE_UTILITY.OLD_CURRENT_SCHEMA	Returns the name of the database schema in the current user environment.
DBE_UTILITY.OLD_CURRENT_USER	Returns the name of the current user.

API	Description
DBE_UTILITY.TABLE_TO_COMMA	Converts a PL/SQL table of names into a comma-delimited string of names.
DBE_UTILITY.GET_SQL_HASH_FUNC	Equivalent to DBE_UTILITY.GET_SQL_HASH. This function is used when **proc_outparam_override** is enabled.
DBE_UTILITY.EXPAND_SQL_TEXT	This is an internal function and is not recommended.
DBE_UTILITY.CANONICALIZE_RET	This is an internal function and is not recommended.
DBE_UTILITY.COMMA_TO_TABLE_FUN	This is an internal function and is not recommended.
DBE_UTILITY.COMPILE_SCH....	This is an internal function and is not recommended. You are advised to use pkg_util.gs_compile_schema.
DBE_UTILITY.NAME_SEPARATE	This is an internal function and is not recommended.
DBE_UTILITY.NAME_TOKENIZE_FUNC	This is an internal function and is not recommended.
DBE_UTILITY.NAME_TOKENIZE_LOWER	This is an internal function and is not recommended.
DBE_UTILITY.NAME_TOKENIZE_LOWER_FUNC	This is an internal function and is not recommended.
DBE_UTILITY.PRIVILEGE_CHECK	This is an internal function and is not recommended.
DBE_UTILITY.SEARCH_CLASS_WITH_NSPOID_ONAME_TYPE	This is an internal function and is not recommended.
DBE_UTILITY.SEARCH_OBJECTS	This is an internal function and is not recommended.
DBE_UTILITY.SEARCH_OBJECTS_SYNONYM_FILL_SECHEMA	This is an internal function and is not recommended.
DBE_UTILITY.SEARCH_PROCEDURE_WITH_NSPOID_ONAME	This is an internal function and is not recommended.
DBE_UTILITY.SEARCH_SYNONM_WITH_NSPOID_ONAME	This is an internal function and is not recommended.
DBE_UTILITY.TABLE_TO_COMMA_FUNC	This is an internal function and is not recommended.
DBE_UTILITY.USER_NAME	This is an internal function and is not recommended.

- DBE_UTILITY.FORMAT_ERROR_BACKTRACE

  Returns the call stack where an error occurs during execution. The prototype of the DBE_UTILITY.FORMAT_ERROR_BACKTRACE function is as follows:

  ```
  DBE_UTILITY.FORMAT_ERROR_BACKTRACE()
  RETURN TEXT;
  ```

- DBE_UTILITY.FORMAT_ERROR_STACK

  Returns the detailed information about the error location when an error occurs during the execution. The prototype of the DBE_UTILITY.FORMAT_ERROR_STACK function is as follows:

  ```
  DBE_UTILITY.FORMAT_ERROR_STACK()
  RETURN TEXT;
  ```

- DBE_UTILITY.FORMAT_CALL_STACK

  Sets the call stack of the output function. The prototype of the DBE_UTILITY.FORMAT_CALL_STACK function is as follows:

  ```
  DBE_UTILITY.FORMAT_CALL_STACK()
  RETURN TEXT;
  ```

- DBE_UTILITY.COMPILE_SCHEMA

  Recompiles the PL/SQL packages and functions (except the packages and functions provided by the system) under a specified schema. The prototype of the DBE_UTILITY.COMPILE_SCHEMA function is as follows:

  ```
  DBE_UTILITY.COMPILE_SCHEMA (
  SCHEMA IN VARCHAR2,
  COMPILE_ALL IN BOOLEAN DEFAULT TRUE,
  REUSE_SETTINGS IN BOOLEAN DEFAULT FALSE
  )
  RETURNS VOID;
  For details about the example, see the usage of the pkg_util.utility_compile_schema function in 11.12.1.2. To call the schema, run the following command:
  call DBE_UTILITY.compile_schema('pkg_var_test');
  ```

- DBE_UTILITY.GET_TIME

  Sets the output time, which is usually used for difference. A separate return value is meaningless. The prototype of the DBE_UTILITY.GET_TIME function is as follows:

  ```
  DBE_UTILITY.GET_TIME()
  RETURN BIGINT;
  ```

- DBE_UTILITY.CANONICALIZE

  Canonicalizes the character string of a table name. The procedure handles a single reserved word or keyword, and removes white spaces for a single identifier so that "table" becomes TABLE. The prototype of the DBE_UTILITY.CANONICALIZE function is as follows:

  ```
  DBE_UTILITY.CANONICALIZE(
     name       IN   VARCHAR2,
     canon_name OUT  VARCHAR2,
     canon_len  IN   BINARY_INTEGER DEFAULT 1024
  );
  ```


Table 10-411 DBE_UTILITY.CANONICALIZE parameters

Parameter	Type	Input/Output Parameter	Whether NULL Is Allowed	Description
name	VARCHAR2	IN	No	Character string to be canonicalized.
canon_name	VARCHAR2	OUT	Yes	Canonicalized character string.
canon_len	BINARY_INTEGER	IN	Yes	Length of the string to be canonicalized. The default value is **1024** (in bytes). If the value of this parameter is less than the actual length (in bytes) of the character string to be standardized, the character string is truncated by byte.

- DBE_UTILITY.COMMA_TO_TABLE

  Converts a comma-delimited string of names into a PL/SQL table of names. The prototype of the DBE_UTILITY.COMMA_TO_TABLE function is as follows:

```
DBE_UTILITY.COMMA_TO_TABLE (
   list   IN  VARCHAR2,
   tablen OUT BINARY_INTEGER,
   tab    OUT VARCHAR2[]
);
```

Table 10-412 DBE_UTILITY.COMMA_TO_TABLE parameters

Parameter	Type	Input/Output Parameter	Whether NULL Is Allowed	Description
list	VARCHAR2	IN	No	A comma-delimited string of names.
tablen	BINARY_INTEGER	OUT	Yes	Number of names in the table.
tab	VARCHAR2	OUT	Yes	Table which contains the string of names.

- DBE_UTILITY.DB_VERSION

  Returns the version number and compatibility version number of the database. The prototype of the DBE_UTILITY.DB_VERSION function is as follows:


```
DBE_UTILITY.DB_VERSION (
   version OUT VARCHAR2
);
```

Table 10-413 DBE_UTILITY.DB_VERSION parameters

Parameter	Type	Input/Output Parameter	Whether NULL Is Allowed	Description
version	VARCHAR2	OUT	No	Output parameter, which indicates the internal database software version. The value is a character string.

- DBE_UTILITY.EXEC_DDL_STATEMENT

  Executes DDL statements entered by users. DBE_UTILITY. The prototype of the EXEC_DDL_STATEMENT function is as follows:

```
DBE_UTILITY. EXEC_DDL_STATEMENT (
   parse_string IN TEXT
);
```

Table 10-414 DBE_UTILITY.EXEC_DDL_STATEMENT parameters

Parameter	Type	Input/Output Parameter	Whether NULL Is Allowed	Description
parse_string	TEXT	IN	Yes	DDL statements to be executed.

- DBE_UTILITY.EXPAND_SQL_TEXT_PROC

  Expands the view of the SQL query. It recursively expands the view objects in the view until a table is displayed. The function prototype of DBE_UTILITY.EXPAND_SQL_TEXT_PROC is as follows:

```
DBE_UTILITY.EXPAND_SQL_TEXT_PROC (
   input_sql_text  IN  CLOB,
   output_sql_text OUT CLOB
);
```

Table 10-415 DBE_UTILITY.EXPAND_SQL_TEXT_PROC parameters

Parameter	Type	Input/Output Parameter	Whether NULL Is Allowed	Description
input_sql_text	CLOB	IN	No	Input SQL text.

Parameter	Type	Input/ Output Parameter	Whether NULL Is Allowed	Description
output_sql_text	CLOB	OUT	No	Output SQL text of the expanded view.

NOTE

In the **input_sql_text** parameter entered by a user, a schema prefix must be added to the object in the SQL statement. If **behavior_compat_options** is set to **bind_procedure_searchpath**, you do not need to forcibly specify the schema prefix.

- DBE_UTILITY.GET_CPU_TIME

  Returns the measured value of the current CPU processing time, in hundredths of a second. The prototype of the DBE_UTILITY.GET_CPU_TIME function is as follows:

```
DBE_UTILITY.GET_CPU_TIME()
RETURN BIGINT;
```

- DBE_UTILITY.GET_ENDIANNESS

  Obtains the big-endian and little-endian information of the byte order on the platform where the database is located. The prototype of the DBE_UTILITY.GET_ENDIANNESS function is as follows:

```
DBE_UTILITY.GET_ENDIANNESS
RETURN INTEGER;
```

- DBE_UTILITY.GET_HASH_VALUE

  Returns the hash value of a given string. The prototype of the DBE_UTILITY.GET_HASH_VALUE function is as follows:

```
DBE_UTILITY.GET_HASH_VALUE(
   name      IN VARCHAR2(n),
   base      IN INTEGER,
   hash_size IN INTEGER)
RETURN INTEGER;
```

Table 10-416 DBE_UTILITY.GET_HASH_VALUE parameters

Parameter	Type	Input/ Output Parameter	Whether NULL Is Allowed	Description
name	VARCHAR2	IN	No	Character string to be hashed.
base	INTEGER	IN	No	Start value of the returned hash value.
hash_size	INTEGER	IN	No	Size of the hash table to which the hash is mapped.

- DBE_UTILITY.GET_SQL_HASH

  Outputs the hash value of a given character string using the MD5 algorithm. The prototype of the DBE_UTILITY.GET_SQL_HASH function is as follows:

```
DBE_UTILITY.GET_SQL_HASH(
   name       IN  VARCHAR2,
   hash       OUT RAW,
   last4bytes OUT BIGINT
);
```

Table 10-417 DBE_UTILITY.GET_SQL_HASH parameters

Parameter	Type	Input/Output Parameter	Whether NULL Is Allowed	Description
name	VARCHAR2	IN	No	Character string to be hashed.
hash	RAW	OUT	No	Complete hexadecimal MD5 hash value.
last4bytes	BIGINT	OUT	No	Last four bytes of the MD5 hash value, which is displayed as an unsigned integer.

NOTE

After setting **set behavior_compat_options** to a value other than **proc_outparam_override** (Contact the administrator for parameter setting.), call the DBE_UTILITY.GET_SQL_HASH function. If DBE_UTILITY.GET_SQL_HASH_FUNC is called, the value assignment fails.

- DBE_UTILITY.IS_BIT_SET

  Checks whether parameter **n** exists in **r**. The prototype of the DBE_UTILITY.IS_BIT_SET function is as follows:

```
DBE_UTILITY.IS_BIT_SET (
   r IN RAW,
   n IN INTEGER)
RETURN INTEGER;
```

Table 10-418 DBE_UTILITY.IS_BIT_SET parameters

Parameter	Type	Input/Output Parameter	Whether NULL Is Allowed	Description
r	RAW	IN	No	4 bytes plus the actual hexadecimal string.
n	INTEGER	IN	No	Determines whether the value exists in the binary system.

- DBE_UTILITY.IS_CLUSTER_DATABASE

  Determines whether the current database is running in database cluster mode. The prototype of the DBE_UTILITY.IS_CLUSTER_DATABASE function is as follows:

```
DBE_UTILITY.IS_CLUSTER_DATABASE
RETURN BOOLEAN;
```

- DBE_UTILITY.NAME_RESOLVE

  Parses the given object name, including synonym translation and necessary authorization checks. The prototype of the DBE_UTILITY.NAME_RESOLVE function is as follows:

```
DBE_UTILITY.NAME_RESOLVE (
   name          IN   VARCHAR2,
   context       IN   INTEGER,
   schema        OUT VARCHAR2,
   part1         OUT VARCHAR2,
   part2         OUT VARCHAR2,
   dblink        OUT VARCHAR2,
   part1_type    OUT INTEGER,
   object_number OUT OID
);
```

Table 10-419 DBE_UTILITY.NAME_RESOLVE parameters

Parameter	Type	Input / Output Parameter	Whether NULL Is Allowed	Description
name	VARCHAR2	IN	No	Name of the object to be parsed. The structure is [[a.]b.]c[@d].
context	INTEGER	IN	No	Start value of the returned hash value.
schema	VARCHAR2	OUT	No	Schema of an object.
part1	VARCHAR2	OUT	No	First part of the name. The type of this column is specified by **part1_type**.
part2	VARCHAR2	OUT	Yes	If this column is not empty, the value is the subprogram name.
dblink	VARCHAR2	OUT	Yes	Database link.

Parameter	Type	Input / Output Parameter	Whether NULL Is Allowed	Description
part1_type	INTEGER	OUT	No	Part 1 types: • **5**: synonym • **7**: procedure (top level) • **8**: function (top level) • **9**: package
object_number	OID	OUT	No	Object ID. In database A, **object_number** is of the numeric type, indicating the object ID. In GaussDB, **object_number** is of the OID type and does not support implicit conversion from a number to an OID.

- DBE_UTILITY.NAME_TOKENIZE

  Parses names in the **a [. b [. c]][@ dblink]** format. If a name contains double quotation marks, the double quotation marks are deleted. Otherwise, the name becomes uppercase letters. The prototype of the DBE_UTILITY.NAME_TOKENIZE function is as follows:

```
DBE_UTILITY.NAME_TOKENIZE (
name    IN  VARCHAR2,
a       OUT VARCHAR2,
b       OUT VARCHAR2,
c       OUT VARCHAR2,
dblink  OUT VARCHAR2,
nextpos OUT INTEGER
);
```

Table 10-420 DBE_UTILITY.NAME_TOKENIZE parameters

Parameter	Type	Input/Output Parameter	Whether NULL Is Allowed	Description
name	VARCHAR2	IN	No	Name, consisting of SQL identifiers (for example, **scott.foo@dblink**).
a	VARCHAR2	OUT	No	First token of the name.
b	VARCHAR2	OUT	Yes	Second token of the name.

Parameter	Type	Input/Output Parameter	Whether NULL Is Allowed	Description
c	VARCHAR2	OUT	Yes	Third token of the name.
dblink	VARCHAR2	OUT	Yes	Database link.
nextpos	INTEGER	OUT	No	Next position of a parsed character string.

- DBE_UTILITY.OLD_CURRENT_SCHEMA

  Returns the name of the database schema in the current user environment. The prototype of the DBE_UTILITY.OLD_CURRENT_SCHEMA function is as follows:

  ```
  DBE_UTILITY.OLD_CURRENT_SCHEMA()
  RETURN VARCHAR;
  ```

- DBE_UTILITY.OLD_CURRENT_USER

  Returns the name of the current user. The prototype of the DBE_UTILITY.OLD_CURRENT_USER function is as follows:

  ```
  DBE_UTILITY.OLD_CURRENT_USER()
  RETURN TEXT;
  ```

- DBE_UTILITY.TABLE_TO_COMMA

  Converts a PL/SQL table of names into a comma-delimited string of names. The prototype of the DBE_UTILITY.TABLE_TO_COMMA function is as follows:

  ```
  DBE_UTILITY.TABLE_TO_COMMA (
     tab    IN  VARCHAR2[],
     tablen OUT BINARY_INTEGER,
     list   OUT VARCHAR2
  );
  ```

  Table 10-421 DBE_UTILITY.TABLE_TO_COMMA parameters

Parameter	Type	Input/Output Parameter	Whether NULL Is Allowed	Description
tab	VARCHAR2[]	IN	No	PL/SQL table which contains the string of names.
tablen	BINARY_INTEGER	OUT	No	Number of names in the PL/SQL table.
list	VARCHAR2	OUT	No	A comma-delimited string of names.

- DBE_UTILITY.GET_SQL_HASH_FUNC

Uses the MD5 algorithm to output the hash value of a given character string. The function prototype of DBE_UTILITY.GET_SQL_HASH_FUNC is:

```
DBE_UTILITY.GET_SQL_HASH_FUNC(
    name       IN  VARCHAR2,
    hash       OUT RAW,
    last4bytes OUT BIGINT
)RETURN BIGINT;
```

Table 10-422 DBE_UTILITY.GET_SQL_HASH_FUNC parameters

Parameter	Type	Input/Output Parameter	Whether NULL Is Allowed	Description
name	VARCHAR2	IN	No	Character string to be hashed.
hash	RAW	OUT	No	Complete hexadecimal MD5 hash value.
last4bytes	BIGINT	OUT	No	Last four bytes of the MD5 hash value, which is displayed as an unsigned integer.

NOTE

After setting **set behavior_compat_options** to **'proc_outparam_override'**, call the DBE_UTILITY.GET_SQL_HASH_FUNC function. If you call the DBE_UTILITY.GET_SQL_HASH function, a parameter mismatch error is reported.

Examples

```
-- Example 1
create or replace procedure print_err() as
DECLARE
    a bool;
BEGIN
    a := not_exist;
exception
    when others then
        dbe_output.print_line('err_stack: ' || DBE_UTILITY.FORMAT_ERROR_STACK());
END;
/
CREATE PROCEDURE

call print_err();
-- The expected result is as follows:
err_stack: 50360452: column "not_exist" does not exist
 print_err
-----------

(1 row)

-- Clean the environment.
drop procedure print_err;
DROP PROCEDURE
```



```
-- Example 2
create or replace procedure print_err() as
DECLARE
    a bool;
BEGIN
    a := not_exist;
exception
    when others then
        dbe_output.print_line('backtrace: ' || DBE_UTILITY.FORMAT_ERROR_BACKTRACE());
END;
/
CREATE PROCEDURE

call print_err();
-- The expected result is as follows:
backtrace: 50360452: PL/pgSQL function print_err() line 5 at assignment

 print_err
-----------

(1 row)

-- Clean the environment.
drop procedure print_err;
DROP PROCEDURE

-- Example 3
create or replace procedure print_err() as
DECLARE
    a bool;
BEGIN
    a := not_exist;
exception
    when others then
        dbe_output.print_line('call_stack: ');
        dbe_output.print_line(DBE_UTILITY.FORMAT_CALL_STACK());
END;
/
CREATE PROCEDURE

call print_err();
-- The expected result is as follows:
call_stack:
      3  dbe_utility.format_call_stack()
      9  print_err()

 print_err
-----------

(1 row)

-- Clean the environment.
drop procedure print_err;
DROP PROCEDURE

-- Example 4
CREATE OR REPLACE PROCEDURE test_get_time1()
AS
declare
    start_time  bigint;
    end_time  bigint;
BEGIN
    start_time:= dbe_utility.get_time ();
    pg_sleep(1);
    end_time:=dbe_utility.get_time ();
    dbe_output.print_line(end_time - start_time);
END;
/
CREATE PROCEDURE
```



```
call test_get_time1();
-- The expected result is as follows:
100
 test_get_time1
----------------

(1 row)

-- Clean the environment.
drop PROCEDURE test_get_time1;
DROP PROCEDURE

-- Example 5
-- Canonicalize the character string of a table name.
declare
    cname varchar2(50);
begin
    dbe_utility.canonicalize('seg1', cname, 50);
    dbe_output.put_line(cname);
end;
/
-- The expected result is as follows:
SEG1
ANONYMOUS BLOCK EXECUTE

-- Example 6
-- Convert the input character string into an array of table names.
DECLARE
 tab_list VARCHAR2(100) := 't1,t2';
 len BINARY_INTEGER;
 tab varchar2[];
BEGIN
 dbe_output.put_line('table list is: ' || tab_list);
 dbe_utility.comma_to_table(tab_list, len, tab);
END;
/
-- The expected result is as follows:
table list is: t1,t2
ANONYMOUS BLOCK EXECUTE

-- Example 7
-- Check the version number and compatibility version number of the database.
declare
    v_version varchar2;
begin
    dbe_utility.db_version(v_version);
    v_version:=left(v_version, 8);
    dbe_output.print_line('version:' || v_version);
end;
/
-- The expected result is as follows:
version:gaussdb
ANONYMOUS BLOCK EXECUTE

-- Example 8
-- Check the measured value of the current CPU processing time.
DECLARE
    cputime NUMBER;
BEGIN
    cputime := dbe_utility.get_cpu_time();
    dbe_output.put_line('cpu time:' || cputime);
END;
/
-- The expected result is as follows (the value is not fixed):
cpu time: 9851
ANONYMOUS BLOCK EXECUTE

-- Example 9
```



```
-- Obtain the big-endian and little-endian information of the byte order on the platform where the
database is located.
BEGIN
    dbe_output.PUT_LINE(dbe_utility.GET_ENDIANNESS());
END;
/
-- The expected result is as follows:
2
ANONYMOUS BLOCK EXECUTE

-- Example 10
-- Obtain the hash value of a given string.
DECLARE
    result NUMBER(28);
BEGIN
    result := dbe_utility.get_hash_value('hello',10,10);
    dbe_output.put_line(result);
END;
/
-- The expected result is as follows:
11
ANONYMOUS BLOCK EXECUTE

-- Example 11
-- Check whether the current database is in cluster mode.
DECLARE
    is_cluster BOOLEAN;
BEGIN
    is_cluster := dbe_utility.IS_CLUSTER_DATABASE();
    dbe_output.put_line('CLUSTER DATABASE: ' || CASE WHEN is_cluster THEN 'TRUE' ELSE 'FALSE' END);
END;
/
-- The expected result is as follows:
CLUSTER DATABASE: FALSE
ANONYMOUS BLOCK EXECUTE

-- Example 12
-- Obtain the name of the database schema in the current user environment.
DECLARE
    schm varchar2(100);
BEGIN
    schm := dbe_utility.old_current_schema();
    dbe_output.put_line('current schema: ' || schm);
END;
/
-- The expected result is as follows (the result is the schema name of the current database, which is not
fixed):
current schema: public
ANONYMOUS BLOCK EXECUTE

-- Example 13
-- Obtain the current username.
select dbe_utility.old_current_user() from sys_dummy;
-- The expected result is as follows (the result is the username of the current database, which is not fixed):
 old_current_user
------------------
 test
(1 row)

-- Example 14
DECLARE
    ddl_str VARCHAR2(255);
BEGIN
    dbe_output.print_line('start to test exec_ddl_statement create table.');
    ddl_str := 'CREATE TABLE test_ddl (COL1 INT)';
    dbe_utility.exec_ddl_statement(ddl_str);
END;
/
-- The expected result is as follows:
```



```
start to test exec_ddl_statement create table.
ANONYMOUS BLOCK EXECUTE

select * from test_ddl;
-- The expected result is as follows:
 col1
------
(0 rows)

-- Clean the environment.
drop table test_ddl;
DROP TABLE

-- Example 15
create table t1 (c1 int primary key, c2 int);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "t1_pkey" for table "t1"
CREATE TABLE
insert into t1 values(1,1),(2,1),(3,2),(4,2),(5,3),(6,3);
INSERT 0 6
create view v1 as select * from t1 where c1 > 1;
CREATE VIEW
create view v2 as select c1 from v1 where c2 > 1;
CREATE VIEW
create view v3 as select * from v2 where c1 > 2;
CREATE VIEW

declare
    in_sql clob := 'select * from public.v3';
    out_sql clob;
begin
    dbe_output.print_line('start to test expand_sql_text_proc v3 expend sql text.');
    dbe_utility.expand_sql_text_proc(in_sql, out_sql);
    dbe_output.print_line(out_sql);
end;
/
-- The expected result is as follows:
start to test expand_sql_text_proc v3 expend sql text.
SELECT c1 FROM (SELECT v2.c1 FROM (SELECT v1.c1 FROM (SELECT t1.c1, t1.c2 FROM public.t1 WHERE
t1.c1 > 1) v1 WHERE v1.c2 > 1) v2 WHERE v2.c1 > 2) v3
ANONYMOUS BLOCK EXECUTE

-- Clean the environment.
drop table t1 cascade;
-- The expected result is as follows:
NOTICE:  drop cascades to 3 other objects
DETAIL:  drop cascades to view v1
drop cascades to view v2
drop cascades to view v3
DROP TABLE

-- Example 16
declare
    name varchar2;
    hash raw;
    last4bytes bigint;
BEGIN
    name := '';
    -- return correctly (D41D8CD98F00B204E9800998ECF8427E, 2118318316)
    dbe_utility.get_sql_hash(name,hash,last4bytes);
    raise notice '%',hash;
    raise notice '%',last4bytes;
END;
/
-- The expected result is as follows:
NOTICE:  D41D8CD98F00B204E9800998ECF8427E
NOTICE:  2118318316
ANONYMOUS BLOCK EXECUTE

-- Example 17
```



```
declare
    bitchar raw(8);
begin
    bitchar := '2111111f';
    dbe_output.print('test '|| bitchar ||' bit is_bit_set value from 1 to 32 bit: ');
    for i in reverse 32 .. 1 loop
        dbe_output.print(dbe_utility.is_bit_set(bitchar, i));
    end loop;
    dbe_output.print_line('.');
end;
/
-- The expected result is as follows:
test 2111111F bit is_bit_set value from 1 to 32 bit: 00100001000100010001000100011111.
ANONYMOUS BLOCK EXECUTE

-- Example 18
create or REPLACE PROCEDURE p_test_pk ( -- for print result
    name in varchar2,
    type in integer
)
as
    schema varchar2;
    part1 varchar2;
    part2 varchar2;
    dblink varchar2;
    part1_type integer;
    object_number integer;
begin
    dbe_utility.name_resolve(name,type,schema,part1,part2,dblink,part1_type,object_number);
    raise notice 'schema: % -- part1: % -- part2: % -- dblink: % -- part1_type: %',
schema,part1,part2,dblink,part1_type;
end;
/
CREATE PROCEDURE

declare begin p_test_pk('a.b.c@aa',3); end;
/
-- The expected result is as follows:
NOTICE:  schema: a -- part1: b -- part2: c -- dblink: aa -- part1_type: 0
CONTEXT:  SQL statement "CALL p_test_pk('a.b.c@aa',3)"
PL/pgSQL function inline_code_block line 1 at PERFORM
ANONYMOUS BLOCK EXECUTE

-- Example 19
DECLARE
    name varchar;
    a    varchar;
    b    varchar;
    c    varchar;
    dblink varchar;
    nextpos INTEGER;
BEGIN
    name := 'Me.w#sdfsdf.CD';
    DBE_UTILITY.NAME_TOKENIZE(name, a, b, c, dblink, nextpos);
    RAISE INFO E'dbe_utility.name_tokenize parse error: name:%\na:%\nb:%\nc:%\ndblink:%',
    name, a, b, c, dblink;
    IF nextpos <> OCTET_LENGTH(name) THEN
        RAISE INFO E'dbe_utility.name_tokenize length error: name:%\nlength of name:%\nnextpos:%',
        name, OCTET_LENGTH(name), nextpos;
    END IF;
END;
/
-- The expected result is as follows:
INFO:  dbe_utility.name_tokenize parse error: name:Me.w#sdfsdf.CD
a: Me
b:W#SDFSDF
c:CD
dblink:<NULL>
ANONYMOUS BLOCK EXECUTE
```



```
-- Example 20
DECLARE
list varchar2(50) := 'aabb,ccdd,eeff,gghh';
len_list integer;
tab varchar2[];
get_list varchar2(50);
len_tab integer;
BEGIN
dbe_output.print_line(''Parameter list:' || list);
dbe_utility.comma_to_table(list,len_list,tab);
dbe_output.print_line('Parameter length:' || len_list);
FOR i IN 1 .. len_list LOOP
    dbe_output.print_line('List name' || i || ' : ' || tab(i));
END LOOP;
dbe_output.print_line('Call table_to_comma:');
dbe_utility.table_to_comma(tab,len_tab,get_list );
dbe_output.print_line('Output:' || get_list );
dbe_output.print_line('Array length:' || len_tab);
END;
/
-- The expected result is as follows:
Parameter list: aabb,ccdd,eeff,gghh
Parameter length: 4
List name 1: aabb
List name 2: ccdd
List name 3: eeff
List name 4: gghh
Call table_to_comma:
Output: aabb,ccdd,eeff,gghh
Array length: 4
ANONYMOUS BLOCK EXECUTE

-- Example 21
declare
    name varchar2;
    hash raw;
    last4bytes bigint;
BEGIN
    name := 'hello world';
    -- return correctly(5EB63BBBE01EEED093CB22BB8F5ACDC3, 3285015183)
    dbe_utility.get_sql_hash_func(name,hash,last4bytes);
    raise notice '%',hash;
    raise notice '%',last4bytes;
END;
/
-- The expected result is as follows:
NOTICE:  3285015183
NOTICE:  <NULL>
ANONYMOUS BLOCK EXECUTE
```

10.12.2.19 DBE_XMLDOM

API Description

The advanced function package DBE_XMLDOM is used to access XMLType objects and implement Document Object Model (DOM), which is an API used to access HTML and XML documents. For details about all types supported by the advanced function package DBE_XMLDOM, see **Table 10-423**. For details about all APIs supported by DBE_XMLDOM, see **Table 10-424**.

NOTE

When the character set of the database is set to **SQL_ASCII** and the **DBE_XMLDOM** advanced package is used, an error message is displayed if the input characters exceed the ASCII range.

Table 10-423 DBE_XMLDOM data types

Type	Description
DOMATTR	Implements the DOMAttributes API.
DOMDOCUMENT	Implements the DOMDocument API.
DOMELEMENT	Implements the DOMElement API.
DOMNAMEDNODEMAP	Implements the DOMNamedNodeMap API.
DOMNODELIST	Implements the DOMNodeList API.
DOMNODE	Implements the DOMNode API.
DOMTEXT	Implements the DOMText API.

Table 10-424 DBE_XMLDOM parameters

API	Description
DBE_XMLDOM.APPENDCHILD	Adds the newchild node to the end of the parent(n) node and returns the newly added node.
DBE_XMLDOM.CREATEELEMENT	Creates a DOMElement object with the specified name.
DBE_XMLDOM.CREATETEXTNODE	Creates a DOMText node.
DBE_XMLDOM.FREEDOCUMENT	Frees resources related to DOMDocument nodes.
DBE_XMLDOM.FREEELEMENT	Frees resources related to DOMElement nodes.
DBE_XMLDOM.FREENODE	Frees resources related to DOMNode nodes.
DBE_XMLDOM.FREENODELIST	Frees resources related to DOMNodeList nodes.
DBE_XMLDOM.GETATTRIBUTE	Returns the attribute values of a DOMElement object by name.
DBE_XMLDOM.GETATTRIBUTES	Returns the attribute values of a DOMNode node as a map.
DBE_XMLDOM.GETCHILDNODES	Converts several subnodes under a node into a node list.
DBE_XMLDOM.GETCHILDRENBYTAGNAME	Returns the subnodes of a DOMElement node by name.

API	Description
DBE_XMLDOM.GETDOCUMENTELEMENT	Returns the first subnode of the specified document.
DBE_XMLDOM.GETFIRSTCHILD	Returns the first subnode.
DBE_XMLDOM.GETLASTCHILD	Returns the last subnode.
DBE_XMLDOM.GETLENGTH	Obtains the number of subnodes under a specified node.
DBE_XMLDOM.GETLOCALNAME	Returns the local name of a node.
DBE_XMLDOM.GETNAMEDITEM	Returns the node specified by name.
DBE_XMLDOM.GETNEXTSIBLING	Returns the next node of the specified node.
DBE_XMLDOM.GETNODENAME	Returns the name of a node.
DBE_XMLDOM.GETNODETYPE	Returns the type of a node.
DBE_XMLDOM.GETNODEVALUE	Obtains the value of a node, depending on its type.
DBE_XMLDOM.GETPARENTNODE	Returns the parent node of a node.
DBE_XMLDOM.GETTAGNAME	Returns the tag name of the specified DOMElement node.
DBE_XMLDOM.HASCHILDNODES	Checks whether the DOMNode object has any subnode.
DBE_XMLDOM.IMPORTNODE	Copies a node and specifies the document to which the node belongs.
DBE_XMLDOM.ISNULL	Checks whether a node is null.
DBE_XMLDOM.ITEM	Returns the item corresponding to the index parameter in the mapping.
DBE_XMLDOM.MAKEELEMENT	Converts a DOMNode object to the DOMElement type.
DBE_XMLDOM.MAKENODE	Forcibly converts a node to the DOMNode type.
DBE_XMLDOM.NEWDOMDOCUMENT	Returns a new DOMDocument object.
DBE_XMLDOM.SETATTRIBUTE	Sets the value of the DOMElement attribute by name.
DBE_XMLDOM.SETCHARSET	Sets the character set for a DOMDocument object.
DBE_XMLDOM.SETDOCTYPE	Sets the external DTD of a DOMDocument object.

API	Description
DBE_XMLDOM.SETNODEVALUE	Sets the value of a node in the DOMNode object.
DBE_XMLDOM.WRITETOBUFFER	Writes an XML node to a specified buffer.
DBE_XMLDOM.WRITETOCLOB	Writes an XML node to a specified CLOB.
DBE_XMLDOM.WRITETOFILE	Writes an XML node to a specified file.
DBE_XMLDOM.GETSESSIONTREENUM	Displays the number of DOM trees of all types in the current session.
DBE_XMLDOM.GETDOCTREESINFO	Displays statistics such as the memory usage and number of nodes of the DOM tree of the document type.
DBE_XMLDOM.GETDETAILDOCTREEINFO	Displays the number of nodes of each type for a specific document variable.
DBE_XMLDOM.GETELEMENTSBYTAGNAME	Returns the list of DOMNodeList nodes that matches TAGNAME.

- DBE_XMLDOM.APPENDCHILD

  Adds the newchild node to the end of the parent(n) node and returns the newly added node. The prototype of the DBE_XMLDOM.APPENDCHILD function is as follows:

```
DBE_XMLDOM.APPENDCHILD(
  n IN DOMNode,
  newchild IN DOMNode)
RETURN DOMNODE;
```

  Table 10-425 DBE_XMLDOM.APPENDCHILD parameters

Parameter	Description
n	Node to be added
newchild	New node added

  NOTE

  1. The error message "operation not support" is displayed for the APPEND ATTR node under the DOCUMENT node. Database A does not report an error in this scenario, but the mounting fails.
  2. The error message "operation not support" is displayed for the APPEND ATTR node under the ATTR node. Database A does not report an error in this scenario, but the mounting fails.
  3. When multiple child nodes of the ATTR type are added to a parent node, the child nodes with the same key value cannot exist under the same parent node.

  Example:


```
-- Add a DOMNode node to a specified DOC tree and use DBE_XMLDOM.HASCHILDNODES() to
check whether the subnode is successfully added.
DECLARE
    doc DBE_XMLDOM.DOMDocument;
    doc1 DBE_XMLDOM.DOMDocument;
    root DBE_XMLDOM.DOMElement;
    rootnode DBE_XMLDOM.DOMNode;
    child1 DBE_XMLDOM.DOMElement;
    child2 DBE_XMLDOM.DOMElement;
    attr DBE_XMLDOM.DOMAttr;
    text DBE_XMLDOM.DOMTEXT;
    node DBE_XMLDOM.DOMNode;
    child1_node DBE_XMLDOM.DOMNode;
    attr_node DBE_XMLDOM.DOMNode;
    parent DBE_XMLDOM.DOMNode;
    buf varchar2(1000);
BEGIN
    doc := DBE_XMLDOM.newDOMDocument();
    root := DBE_XMLDOM.createElement(doc, 'root');
    rootnode := DBE_xmldom.makeNode(root);
    node := DBE_XMLDOM.appendChild(DBE_xmldom.makeNode(doc), rootnode);
    child1 := DBE_XMLDOM.createElement(doc, 'child1');
    child1_node := DBE_XMLDOM.makeNode(child1);
    node := DBE_XMLDOM.appendChild(rootnode, child1_node);
    attr := DBE_XMLDOM.createAttribute(doc, 'abc');
    attr_node := DBE_XMLDOM.makeNode(attr);
    node := DBE_XMLDOM.appendChild(child1_node, attr_node);
    IF DBE_XMLDOM.HASCHILDNODES(child1_node) THEN
        DBE_OUTPUT.print_line('HAS CHILD NODES');
    ELSE
        DBE_OUTPUT.print_line('NOT HAS CHILD NODES ');
    END IF;
    parent := DBE_XMLDOM.GETPARENTNODE(attr_node);
    buf := DBE_XMLDOM.GETNODENAME(parent);
    DBE_OUTPUT.print_line(buf);
END;
/
-- Expected result:
NOT HAS CHILD NODES
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.CREATEELEMENT

  Returns the DOMElement object with the specified name. The prototype of the DBE_XMLDOM.CREATEELEMENT function is as follows:

```
DBE_XMLDOM.CREATEELEMENT(
  doc        IN      DOMDOCUMENT,
  tagName    IN      VARCHAR2)
 RETURN DOMELEMENT;
```

  Returns the DOMElement object with the specified name and namespace. The prototype of the DBE_XMLDOM.CREATEELEMENT function is as follows:

```
DBE_XMLDOM.CREATEELEMENT(
  doc        IN     DOMDOCUMENT,
  tagName    IN     VARCHAR2,
  ns         IN     VARCHAR2)
 RETURN DOMELEMENT;
```

Table 10-426 DBE_XMLDOM.CREATEELEMENT parameters

Parameter	Description
doc	Specified DOMDocument object
tagName	Name of the new DOMElement object

Parameter	Description
ns	Namespace

NOTE

1. When the **tagName** parameter is set to null or an empty character string, the exception "NULL or invalid TagName argument specified" is thrown.
2. The default maximum length of **tagName** and **ns** is 32767. If the length exceeds 32767, an exception is thrown.

Example:

```
-- 1. Create a DOMElement object with the specified name.
DECLARE
  doc dbe_xmldom.domdocument;
  attr DBE_XMLDOM.DOMATTR;
  elem DBE_XMLDOM.DOMELEMENT;
  ans DBE_XMLDOM.DOMATTR;
  buf varchar2(1010);
BEGIN
   doc := dbe_xmldom.newdomdocument('<?xml version="1.0" encoding="UTF-8"?>
      <computer size="ITX"><cpu>Ryzen 9 3950X</cpu>
      <ram>32GBx2 DDR4 3200MHz</ram>
      <motherboard>ROG X570i</motherboard>
      <gpu>RTX2070 Super</gpu>
      <ssd>1TB NVMe Toshiba + 2TB NVMe WD Black</ssd>
      <hdd>12TB WD Digital</hdd>
      <psu>CORSAIR SF750</psu>
      <case>LIANLI TU150</case>
      </computer>');
   elem := dbe_xmldom.createelement(doc,'elem');
   DBE_XMLDOM.WRITETOBUFFER(dbe_xmldom.makenode(elem), buf);
   DBE_OUTPUT.print_line(buf);
END;
/
-- Expected result:
ANONYMOUS BLOCK EXECUTE

-- 2. Create a DOMElement object with the specified name and namespace.
DECLARE
  doc dbe_xmldom.domdocument;
  attr DBE_XMLDOM.DOMATTR;
  elem DBE_XMLDOM.DOMELEMENT;
  ans DBE_XMLDOM.DOMNODE;
  buf varchar2(1010);
  list DBE_XMLDOM.DOMNODELIST;
  node DBE_XMLDOM.DOMNODE;
BEGIN
   doc := dbe_xmldom.newdomdocument('<h:data xmlns:h="http://www.w3.org/TR/html4/">
      <h:da1 len="10">test namespace</h:da1><h:da1>bbbbbbbbbb</h:da1></h:data>');
   elem := dbe_xmldom.createelement(doc,'elem','http://www.w3.org/TR/html5/');
   ans := DBE_XMLDOM.APPENDCHILD(dbe_xmldom.makenode(doc), dbe_xmldom.makenode(elem));
   DBE_XMLDOM.WRITETOBUFFER(doc, buf);
   DBE_OUTPUT.print_line(buf);
END;
/
-- Expected result:
<?xml version="1.0" encoding="UTF-8"?>
<h:data xmlns:h="http://www.w3.org/TR/html4/">
 <h:da1 len="10">test namespace</h:da1>
 <h:da1>bbbbbbbbbb</h:da1>
</h:data>
<elem xmlns="http://www.w3.org/TR/html5/"/>

ANONYMOUS BLOCK EXECUTE
```


- DBE_XMLDOM.CREATETEXTNODE

  Creates and returns a DOMText object. The prototype of the DBE_XMLDOM.CREATETEXTNOD function is as follows:

```
DBE_XMLDOM.CREATETEXTNODE(
  doc IN DOMDocument,
  data IN VARCHAR2)
RETURN DOMTEXT;
```

Table 10-427 DBE_XMLDOM.CREATETEXTNODE parameters

Parameter	Description
doc	Specified DOMDocument object
data	Content of the DOMText node

NOTE

1. You can enter an empty string or null value for **data**.
2. The default maximum length of **data** is 32767. If the length exceeds 32767, an exception is thrown.

Example:

```
-- Add a DOMText node to the DOC tree and print the DOC tree to the buffer.
DECLARE
  doc DBE_XMLDOM.DOMDOCUMENT;
  doctext DBE_XMLDOM.DOMTEXT;
  node DBE_XMLDOM.DOMNODE;
  buffer varchar2(1010);
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
     <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)>
     <!ELEMENT to (#PCDATA)>
     <!ELEMENT from (#PCDATA)>
     <!ELEMENT heading (#PCDATA)>
     <!ELEMENT body (#PCDATA)>]>
     <note>
       <to>Chinese</to>
       <from>Jani</from>
       <heading>Reminder</heading>
       <body>Don''t forget me this weekend!</body>
     </note>');
  doctext := DBE_XMLDOM.CREATETEXTNODE(doc, 'there is nothing');
  node := DBE_XMLDOM.MAKENODE(doctext);
  dbe_xmldom.writetobuffer(node, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
END;
/
-- Expected result:
buffer:
there is nothing
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.FREEDOCUMENT

  Frees a DOMDocument node. The prototype of the DBE_XMLDOM.FREEDOCUMENT function is as follows:

```
DBE_XMLDOM.FREEDOCUMENT(
  doc    IN    DOMDOCUMENT);
```


Table 10-428 DBE_XMLDOM.FREEDOCUMENT parameters

Parameter	Description
doc	Specified DOMDocument node

Example:

```
-- Free the entire DOC tree after the DOMNode node is added to the DOC tree.
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  doc_node dbe_xmldom.DOMNODE;
  root_elmt dbe_xmldom.DOMELEMENT;
  root_node dbe_xmldom.DOMNODE;
  value  varchar(1000);
BEGIN
  doc := dbe_xmldom.newdomdocument();
  doc_node := dbe_xmldom.MAKENODE(doc);
  root_elmt := dbe_xmldom.CREATEELEMENT(doc,'staff');
  root_node:=dbe_xmldom.APPENDCHILD(doc_node, dbe_xmldom.MAKENODE(root_elmt));
  dbe_xmldom.freedocument(doc);
END;
/
-- Expected result:
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.FREEELEMENT

  Frees a DOMElement node. The prototype of the DBE_XMLDOM.FREEELEMENT function is as follows:

```
DBE_XMLDOM.FREEELEMENT(
  elem     IN     DOMELEMENT);
```

Table 10-429 DBE_XMLDOM.FREEELEMENT parameters

Parameter	Description
elem	Specified DOMElement node

Example:

```
-- Obtain a DOMElement node from the DOC tree and free the node. Check whether the node is
empty before and after the node is freed.
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  node dbe_xmldom.domnode;
  node1 dbe_xmldom.domnode;
  nodelist DBE_XMLDOM.DOMNODELIST;
  len INTEGER;
  buffer varchar2(1010);
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
     <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)>
     <!ELEMENT to (#PCDATA)>
     <!ELEMENT from (#PCDATA)>
     <!ELEMENT heading (#PCDATA)>
     <!ELEMENT body (#PCDATA)>]>
     <note>
       <to>Chinese</to>
       <from>Jani</from>
```



```
        <heading>Reminder</heading>
        <body>Don''t forget me this weekend!</body>
      </note>');
  elem := dbe_xmldom.GETDOCUMENTELEMENT(doc);
  IF DBE_XMLDOM.ISNULL(elem) THEN
      dbe_output.print_line('IS NULL');
  ELSE
      dbe_output.print_line('NOT NULL');
  END IF;
  dbe_xmldom.FREEELEMENT(elem);
  IF DBE_XMLDOM.ISNULL(elem) THEN
       dbe_output.print_line('IS NULL');
  ELSE
       dbe_output.print_line('NOT NULL');
  END IF;
END;
/
-- Expected result:
NOT NULL
IS NULL
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.FREENODE

  Frees a DOMNode node. The prototype of the DBE_XMLDOM.FREENODE function is as follows:

```
DBE_XMLDOM.FREENODE(
  n IN DOMNODE);
```

Table 10-430 DBE_XMLDOM.FREENODE parameters

Parameter	Description
n	Specified DOMNode node

NOTE

1. After GaussDB performs the FREENODE operation, the freed node is not available again. After the database A performs the FREENODE operation, the freed node is available again and becomes another node.
2. When other APIs call the freed DOMNode node, the calling is different from that in database A.

Example:

```
-- Obtain a DOMNode node from the DOC tree and free the node. Check whether the node is empty
before and after the node is freed.
DECLARE
  doc dbe_xmldom.domdocument;
  node dbe_xmldom.domnode;
  node1 dbe_xmldom.domnode;
  nodelist DBE_XMLDOM.DOMNODELIST;
  len INTEGER;
  buffer1 varchar2(1010);
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
     <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)>
     <!ELEMENT to (#PCDATA)>
     <!ELEMENT from (#PCDATA)>
     <!ELEMENT heading (#PCDATA)>
     <!ELEMENT body (#PCDATA)>]>
     <note>
       <to>Chinese</to>
       <from>Jani</from>
```



```
        <heading>Reminder</heading>
        <body>Don''t forget me this weekend!</body>
      </note>');
  node := dbe_xmldom.makenode(doc);
  node := dbe_xmldom.GETFIRSTCHILD(node);
  IF DBE_XMLDOM.ISNULL(node) THEN
        dbe_output.print_line('IS NULL');
  ELSE
        dbe_output.print_line('NOT NULL');
  END IF;
  DBE_XMLDOM.FREENODE(node);
  IF DBE_XMLDOM.ISNULL(node) THEN
        dbe_output.print_line('IS NULL');
  ELSE
        dbe_output.print_line('NOT NULL');
  END IF;
END;
/
-- Expected result:
NOT NULL
IS NULL
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.FREENODELIST

  Frees a DOMNodeList node. The prototype of the DBE_XMLDOM.FREENODE function is as follows:

```
DBE_XMLDOM.GETLENGTH(
  nl     IN    DOMNODELIST);
```

Table 10-431 DBE_XMLDOM.FREENODELIST parameters

Parameter	Description
nl	Specified DOMNodeList node

NOTE

1. A DOMNodeList node will be completed freed by FREENODELIST.
2. When other APIs call the freed DOMNodelist node, the calling is different from that in database A.
3. The input parameter **freenodelist** cannot be empty.

Example:

```
-- Obtain a DOMNodeList node from the DOC tree and free the node. Check the node length before and after the node is freed.
DECLARE
  doc dbe_xmldom.domdocument;
  node dbe_xmldom.domnode;
  node1 dbe_xmldom.domnode;
  nodelist DBE_XMLDOM.DOMNODELIST;
  len INTEGER;
  buffer1 varchar2(1010);
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
     <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)>
     <!ELEMENT to (#PCDATA)>
     <!ELEMENT from (#PCDATA)>
     <!ELEMENT heading (#PCDATA)>
     <!ELEMENT body (#PCDATA)>]>
     <note>
       <to>Chinese</to>
       <from>Jani</from>
```



```
        <heading>Reminder</heading>
        <body>Don''t forget me this weekend!</body>
      </note>');
  node := dbe_xmldom.makenode(doc);
  node := dbe_xmldom.GETFIRSTCHILD(node);
  nodelist := DBE_XMLDOM.GETCHILDNODES(node);
  len := DBE_XMLDOM.GETLENGTH(nodelist);
  RAISE NOTICE 'len :  %', len;
  DBE_XMLDOM.FREENODELIST(nodelist);
  len := DBE_XMLDOM.GETLENGTH(nodelist);
  RAISE NOTICE 'len :  %', len;
END;
/
-- Expected result:
NOTICE:  len :  4
NOTICE:  len :  0
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.GETATTRIBUTE

  Returns the attribute values of a DOMElement object by name. The prototype of the DBE_XMLDOM.GETATTRIBUTE function is as follows:

```
DBE_XMLDOM.GETATTRIBUTE(
  elem       IN      DOMELEMENT,
  name       IN      VARCHAR2)
RETURN VARCHAR2;
```

  Returns the attribute values of a DOMElement object by name and namespace URI. The prototype of the DBE_XMLDOM.GETATTRIBUTE function is as follows:

```
DBE_XMLDOM.GETATTRIBUTE(
  elem      IN     DOMELEMENT,
  name      IN     VARCHAR2,
  ns        IN     VARCHAR2)
RETURN VARCHAR2;
```

Table 10-432 DBE_XMLDOM.GETATTRIBUTE parameters

Parameter	Description
elem	Specified DOMElement node
name	Attribute name
ns	Namespace

NOTE

1. The **ns** parameter of the DBE_XMLDOM.GETATTRIBUTE API does not support the asterisk (*) parameter.
2. GaussDB does not support the namespace prefix as an attribute, and the value of the prefix cannot be queried through the DBE_XMLDOM.GETATTRIBUTE API.

Example:

```
-- 1. Return the attribute values of a DOMElement object by name.
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  docnode DBE_XMLDOM.DOMNode;
  buffer varchar2(1010);
  value  varchar2(1000);
BEGIN
```



```
  doc := dbe_xmldom.newDOMDocument();
  elem := DBE_XMLDOM.CREATEELEMENT(doc, 'root');
  DBE_XMLDOM.setattribute(elem, 'len', '50cm');
  docnode := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(doc),
DBE_XMLDOM.makeNode(elem));
  value := DBE_XMLDOM.getattribute(elem, 'len');
  dbe_output.print_line('value: ');
  dbe_output.print_line(value);
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
END;
/
-- Expected result:
value:
50cm
buffer:
<?xml version="1.0" encoding="UTF-8"?>
<root len="50cm"/>

ANONYMOUS BLOCK EXECUTE

-- 2. Return the attribute values of a DOMElement object by name and namespace URI.
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  docnode DBE_XMLDOM.DOMNode;
  buffer varchar2(1010);
  value  varchar(1000);

BEGIN
  doc := dbe_xmldom.newDOMDocument();
  elem := DBE_XMLDOM.CREATEELEMENT(doc, 'root');
  DBE_XMLDOM.setattribute(elem, 'len', '50cm', 'www.huawei.com');
  docnode := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(doc),
DBE_XMLDOM.makeNode(elem));
  value := DBE_XMLDOM.getattribute(elem, 'len', 'www.huawei.com');
  dbe_output.print_line('value: ');
  dbe_output.print_line(value);
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
END;
/
-- Expected result:
value:
50cm
buffer:
<?xml version="1.0" encoding="UTF-8"?>
<root len="50cm"/>

ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.GETATTRIBUTES

  Returns the attribute values of a DOMNode node as a map. The prototype of the DBE_XMLDOM.GETATTRIBUTES function is as follows:

```
DBE_XMLDOM.GETATTRIBUTES(
  n IN DOMNode)
RETURN DOMNAMEDNODEMAP;
```

Table 10-433 DBE_XMLDOM.GETATTRIBUTES parameters

Parameter	Description
n	Specified DOMNode node

Example:

```
-- Obtain the attribute values of a DOMNode node, return an object of the DOMNamedNodeMap
type, and output the length of the DOMNamedNodeMap object and the value of the first node.
DECLARE
  doc dbe_xmldom.domdocument;
  node dbe_xmldom.domnode;
  node1 dbe_xmldom.domnode;
  len INTEGER;
  map DBE_XMLDOM.DOMNAMEDNODEMAP;
  buffer1 varchar2(1010);
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
     <note a="16" b="176" c="asd">
       <to>Chinese</to>
       <from>Jani</from>
       <heading>Reminder</heading>
       <body>Don''t forget me this weekend!</body>
     </note>');
  node := dbe_xmldom.makenode(doc);
  node := dbe_xmldom.GETFIRSTCHILD(node);
  map := DBE_XMLDOM.GETATTRIBUTES(node);
  IF DBE_XMLDOM.ISNULL(map) THEN
        dbe_output.print_line('IS NULL');
  ELSE
        dbe_output.print_line('NOT NULL');
  END IF;
    len := DBE_XMLDOM.GETLENGTH(map);
  RAISE NOTICE 'len :  %', len;
   node1 := DBE_XMLDOM.ITEM(map, 0);
  dbe_xmldom.writetobuffer(node1, buffer1);
  dbe_output.print_line('buffer1: ');
  dbe_output.print_line(buffer1);
END;
/
-- Expected result:
NOT NULL
NOTICE:  len :  3
buffer1:
16
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.GETCHILDNODES

  Converts several subnodes under a node into a node list. The prototype of the DBE_XMLDOM.GETCHILDNODES function is as follows:

```
DBE_XMLDOM.GETCHILDNODES(
  n IN DOMNode)
RETURN DOMNodeList;
```

Table 10-434 DBE_XMLDOM.GETCHILDNODES parameters

Parameter	Description
n	Specified DOMNode node

Example:

```
-- After obtaining the first subnode of the DOC tree, convert several subnodes under the node into a
node list and output the length information.
DECLARE
   doc dbe_xmldom.domdocument;
   doc_node dbe_xmldom.domnode;
   root_node dbe_xmldom.domnode;
   node_list dbe_xmldom.domnodelist;
   list_len integer;
```



```
  node_name varchar2(1000);
  node_type integer;
  buffer varchar2(1010);
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
    <note>
      <to>Chinese</to>
      <from>Jani</from>
      <heading>Reminder</heading>
      <body>Don''t forget me this weekend!</body>
    </note>');
  doc_node := DBE_XMLDOM.MAKENODE(doc);
  root_node := DBE_XMLDOM.GETFIRSTCHILD(doc_node);
  node_name := DBE_XMLDOM.GETNODENAME(root_node);
  node_type := DBE_XMLDOM.GETNODETYPE(root_node);
  dbe_output.print_line(node_name);
  dbe_output.print_line(node_type);
  node_list := DBE_XMLDOM.GETCHILDNODES(root_node);
  list_len := DBE_XMLDOM.GETLENGTH(node_list);
  dbe_output.print_line(list_len);
END;
/
-- Expected result:
note
1
4
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.GETCHILDRENBYTAGNAME

  Returns the subnodes of a DOMElement node by name. The prototype of the DBE_XMLDOM.GETCHILDRENBYTAGNAME function is as follows:

```
DBE_XMLDOM.GETCHILDRENBYTAGNAME (
  elem       IN      DOMELEMENT,
  name       IN      VARCHAR2)
RETURN DOMNODELIST;
```

  Returns the subnodes of a DOMElement node by name and namespace. The prototype of the DBE_XMLDOM.GETCHILDRENBYTAGNAME function is as follows:

```
DBE_XMLDOM.GETCHILDRENBYTAGNAME (
  elem      IN     DOMELEMENT,
  name      IN     VARCHAR2,
  ns        IN     VARCHAR2)
 RETURN DOMNODELIST;
```

  Table 10-435 DBE_XMLDOM.GETCHILDRENBYTAGNAME parameters

Parameter	Description
elem	Specified DOMElement node
name	Attribute name
ns	Namespace

  NOTE

  The **ns** parameter of the DBE_XMLDOM.GETCHILDRENBYTAGNAME API does not support the asterisk (*) parameter. To obtain all attributes of a node, use the DBE_XMLDOM.GETCHILDNODES API.

  Example:


```
-- 1. Return the subnodes of a DOMElement node by name.
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  docnodelist dbe_xmldom.domnodelist;
  node_elem dbe_xmldom.domelement;
  node dbe_xmldom.domnode;
  buffer varchar2(1010);
  value  varchar2(1000);
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0" encoding="UTF-8"?>
     <students age="16" hight="176">
     <student>
       <name>Jerry</name><age>519</age><sex>man</sex><abc>12345</abc>
     </student>
     <student>
       <name>Bob</name><age>245</age><sex>woman</sex><abc>54321</abc>
     </student>
     </students>');
  elem := dbe_xmldom.GETDOCUMENTELEMENT(doc);
  docnodelist := dbe_xmldom.GETCHILDRENBYTAGNAME(elem, 'student');
  node := dbe_xmldom.ITEM(docnodelist, 0);
  node_elem := dbe_xmldom.makeelement(node);
  value := DBE_XMLDOM.gettagname(node_elem);
  dbe_output.print_line('value: ');
  dbe_output.print_line(value);
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
END;
/
-- Expected result:
value:
student
buffer:
<?xml version="1.0" encoding="UTF-8"?>
<students age="16" hight="176">
 <student>
  <name>Jerry</name>
  <age>519</age>
  <sex>man</sex>
  <abc>12345</abc>
 </student>
 <student>
  <name>Bob</name>
  <age>245</age>
  <sex>woman</sex>
  <abc>54321</abc>
 </student>
</students>

ANONYMOUS BLOCK EXECUTE

-- 2. Return the child nodes of DOMElement by name and namespace.
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  node dbe_xmldom.domnode;
  node_elem dbe_xmldom.domelement;
  docnodelist dbe_xmldom.domnodelist;
  buffer varchar2(1010);
  value varchar2(1000);
BEGIN
  doc := dbe_xmldom.newdomdocument('
    <note xmlns:h="www.huawei.com">
    <h:to h:len="50cm">Chinese</h:to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don''t forget me this weekend!</body>
```



```
    </note>');
  elem := dbe_xmldom.GETDOCUMENTELEMENT(doc);
  docnodelist := dbe_xmldom.GETCHILDRENBYTAGNAME(elem, 'to', 'www.huawei.com');
  node := dbe_xmldom.ITEM(docnodelist, 0);
  node_elem := dbe_xmldom.makeelement(node);
  value := DBE_XMLDOM.getattribute(node_elem, 'len');
  dbe_output.print_line('value: ');
  dbe_output.print_line(value);
END;
/
-- Expected result:
value:
50cm
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.GETDOCUMENTELEMENT

  Returns the first subnode of the specified document. The prototype of the DBE_XMLDOM.GETDOCUMENTELEMENT function is as follows:

```
DBE_XMLDOM.GETDOCUMENTELEMENT(
  doc      IN      DOMDOCUMENT)
 RETURN DOMELEMENT;
```

  Table 10-436 DBE_XMLDOM.GETDOCUMENTELEMENT parameters

Parameter	Description
doc	Specified DOMDocument node

  Example:

```
-- Obtain the first subnode in the DOC tree and output the node name.
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  doc_node dbe_xmldom.DOMNODE;
  root_elmt dbe_xmldom.DOMELEMENT;
  root_node dbe_xmldom.DOMNODE;
  value  varchar(1000);
BEGIN
  doc := dbe_xmldom.newdomdocument();
  doc_node := dbe_xmldom.MAKENODE(doc);
  root_elmt := dbe_xmldom.CREATEELEMENT(doc,'staff');
  root_node:=dbe_xmldom.APPENDCHILD(doc_node, dbe_xmldom.MAKENODE(root_elmt));
  elem := dbe_xmldom.GETDOCUMENTELEMENT(doc);
  value := DBE_XMLDOM.gettagname(elem);
  dbe_output.print_line(value);
END;
/
-- Expected result:
staff
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.GETFIRSTCHILD

  Returns the first subnode of a node. The prototype of the DBE_XMLDOM.GETFIRSTCHILD function is as follows:

```
DBE_XMLDOM.GETFIRSTCHILD(
  n IN DOMNODE)
RETURN DOMNODE;
```


Table 10-437 DBE_XMLDOM.GETFIRSTCHILD parameters

Parameter	Description
n	Specified DOMNode node

Example:

```
-- Obtain the name and type of the first subnode after the DOC tree is converted to the DOMNode type, and then obtain the name of the first subnode of the obtained first DOMNode node.
DECLARE
    doc dbe_xmldom.domdocument;
    doc_node dbe_xmldom.domnode;
     root_node dbe_xmldom.domnode;
    inside_node dbe_xmldom.domnode;
    node_name varchar2(1000);
    node_type integer;
BEGIN
    doc := dbe_xmldom.newdomdocument('<?xml version="1.0" encoding="UTF-8"?>
       <students age="16" hight="176">
       <student1>
          <name>Jerry</name><age>519</age><sex>man</sex><abc>12345</abc>
       </student1>
       <student2>
          <name>Bob</name><age>245</age><sex>woman</sex><abc>54321</abc>
       </student2>
       </students>');
    doc_node := DBE_XMLDOM.MAKENODE(doc);
    root_node := DBE_XMLDOM.GETFIRSTCHILD(doc_node);
    node_name := DBE_XMLDOM.GETNODENAME(root_node);
    node_type := DBE_XMLDOM.GETNODETYPE(root_node);
    dbe_output.print_line(node_name);
    dbe_output.print_line(node_type);
    inside_node := DBE_XMLDOM.GETFIRSTCHILD(root_node);
    node_name := DBE_XMLDOM.GETNODENAME(inside_node);
    dbe_output.print_line(node_name);
END;
/
-- Expected result:
students
1
student1
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.GETLASTCHILD

  Returns the last subnode of a node. The prototype of the DBE_XMLDOM.GETLASTCHILD function is as follows:

```
DBE_XMLDOM.GETLASTCHILD(
  n IN DOMNODE)
RETURN DOMNODE;
```

Table 10-438 DBE_XMLDOM.GETLASTCHILD parameters

Parameter	Description
n	Specified DOMNode node

Example:

```
-- Obtain the name and type of the last subnode after the DOC tree is converted to the DOMNode type, and then obtain the name of the last subnode of the obtained last DOMNode node.
```



```
DECLARE
    doc dbe_xmldom.domdocument;
    doc_node dbe_xmldom.domnode;
    root_node dbe_xmldom.domnode;
    inside_node dbe_xmldom.domnode;
    node_name varchar2(1000);
    node_type integer;
BEGIN
    doc := dbe_xmldom.newdomdocument('<?xml version="1.0" encoding="UTF-8"?>
      <students age="16" hight="176">
      <student1>
        <name>Jerry</name><age>519</age><sex>man</sex><abc>12345</abc>
      </student1>
      <student2>
        <name>Bob</name><age>245</age><sex>woman</sex><abc>54321</abc>
      </student2>
      </students>');
    doc_node := DBE_XMLDOM.MAKENODE(doc);
    root_node := DBE_XMLDOM.GETFIRSTCHILD(doc_node);
    node_name := DBE_XMLDOM.GETNODENAME(root_node);
    node_type := DBE_XMLDOM.GETNODETYPE(root_node);
    dbe_output.print_line(node_name);
    dbe_output.print_line(node_type);
    inside_node := DBE_XMLDOM.GETLASTCHILD(root_node);
    node_name := DBE_XMLDOM.GETNODENAME(inside_node);
    dbe_output.print_line(node_name);
END;
/
-- Expected result:
students
1
student2
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.GETLENGTH

  Returns the number of subnodes under a DOMNamedNodeMap node. The prototype of the DBE_XMLDOM.GETLENGTH function is as follows:

  ```
  DBE_XMLDOM.GETLENGTH(
    nnm      IN     DOMNAMEDNODEMAP)
   RETURN NUMBER;
  ```

  Returns the number of subnodes under a DOMNodeList node. The prototype of the DBE_XMLDOM.GETLENGTH function is as follows:

  ```
  DBE_XMLDOM.GETLENGTH(
    nl     IN    DOMNODELIST)
   RETURN NUMBER;
  ```

  Table 10-439 DBE_XMLDOM.GETLENGTH parameters

Parameter	Description
nnm	Specified DOMNamedNodeMap node
nl	Specified DOMNodeList node

  Example:

  ```
  -- 1. Declare a DOMNamedNodeMap parameter in a function.
  DECLARE
    doc DBE_XMLDOM.DOMDocument;
    elem DBE_XMLDOM.DOMElement;
    map DBE_XMLDOM.DOMNAMEDNODEMAP;
    node DBE_XMLDOM.DOMNODE;
  ```



```
  buf varchar2(10000);
  len INTEGER;
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
     <bookstore category="web" cover="paperback">
     <book category="cooking">
       <title lang="en">Everyday Italian</title>
       <author>Giada De Laurentiis</author>
       <year>2005</year>
       <price>30.00</price>
     </book>
     </bookstore>');
  elem := DBE_XMLDOM.GETDOCUMENTELEMENT(doc);
  node := DBE_XMLDOM.MAKENODE(elem);
  map := DBE_XMLDOM.GETATTRIBUTES(node);
  len := DBE_XMLDOM.GETLENGTH(map);
  DBE_OUTPUT.print_line(len);
END;
/
-- Expected result:
2
ANONYMOUS BLOCK EXECUTE

-- 2. Declare a NodeList parameter in a function.
DECLARE
  doc dbe_xmldom.domdocument;
  node dbe_xmldom.domnode;
  node1 dbe_xmldom.domnode;
  nodelist DBE_XMLDOM.DOMNODELIST;
  len INTEGER;
  buffer1 varchar2(1010);
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0" encoding="UTF-8"?>
     <students age="16" hight="176">
     <student>
       <name>Jerry</name><age>519</age><sex>man</sex><abc>12345</abc>
     </student>
     <student>
       <name>Jerry</name><age>519</age><sex>man</sex><abc>12345</abc>
     </student>
     </students>');
  node := dbe_xmldom.makenode(doc);
  node := dbe_xmldom.GETFIRSTCHILD(node);
  nodelist := DBE_XMLDOM.GETCHILDNODES(node);
  len := DBE_XMLDOM.GETLENGTH(nodelist);
  RAISE NOTICE 'len :  %', len;
END;
/
-- Expected result:
NOTICE:  len :  2
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.GETLOCALNAME

  Returns the local name of the given DOMAttr node. The prototype of the DBE_XMLDOM.MAKENODE function is as follows:

  ```
  DBE_XMLDOM.GETLOCALNAME(
    a       IN     DOMATTR)
  RETURN VARCHAR2;
  ```

  Returns the local name of the given DOMElement node. The prototype of the DBE_XMLDOM.MAKENODE function is as follows:

  ```
  DBE_XMLDOM.GETLOCALNAME(
    elem       IN     DOMELEMENT)
  RETURN VARCHAR2;
  ```

  Returns the local name of the given DOMNode node. The prototype of the DBE_XMLDOM.MAKENODE function is as follows:


```
DBE_XMLDOM.GETLOCALNAME(
  n      IN     DOMNODE,
  data   OUT    VARCHAR2);
```

Table 10-440 DBE_XMLDOM.GETLOCALNAME parameters

Parameter	Description
a	Specified DOMAttr node
elem	Specified DOMElement node
n	Specified DOMNode node
data	Returned local name

Example:

```
-- 1. Use the createAttribute function to generate an attr node to obtain the local name.
DECLARE
   doc DBE_XMLDOM.DOMDocument;
   root DBE_XMLDOM.DOMElement;
   attr1 DBE_XMLDOM.DOMATTR;
   value VARCHAR2(1000);
BEGIN
   doc := DBE_xmldom.newdomdocument('<?xml version="1.0"?>
      <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)>
      <!ELEMENT to (#PCDATA)>
      <!ELEMENT from (#PCDATA)>
      <!ELEMENT heading (#PCDATA)>
      <!ELEMENT body (#PCDATA)>]>
      <note><to>Chinese</to>
      <from>Jani</from>
      <heading>Reminder</heading>
      <body>Don''t forget me this weekend!</body>
      </note>');
   attr1 := DBE_XMLDOM.createAttribute(doc,'len');
   value := DBE_XMLDOM.getlocalname(attr1);
   DBE_output.print_line('value: ');
   DBE_output.print_line(value);
END;
/
-- Expected result:
value:
len
ANONYMOUS BLOCK EXECUTE

-- 2. Use the createElement function to generate a DOMElement node and obtain its local name.
DECLARE
   doc DBE_xmldom.domdocument;
   elem DBE_xmldom.domelement;
   value  varchar2(10000);
BEGIN
   doc := DBE_xmldom.newdomdocument();
   elem := DBE_XMLDOM.createELEMENT(doc, 'root');
   value := DBE_XMLDOM.getlocalname(elem);
   DBE_output.print_line('value: ');
   DBE_output.print_line(value);
END;
/
-- Expected result:
value:
root
ANONYMOUS BLOCK EXECUTE

-- 3. Convert a DOMElement to a DOMNode node and obtain its local name.
```



```
DECLARE
  doc DBE_xmldom.domdocument;
  elem DBE_xmldom.domelement;
  node DBE_xmldom.domnode;
  value  varchar2(100);
  buf varchar2(100);
BEGIN
  doc := DBE_xmldom.newdomdocument();
  elem := DBE_XMLDOM.createELEMENT(doc, 'root');
  node := DBE_xmldom.makenode(elem);
  DBE_XMLDOM.getlocalname(node, buf);
  DBE_output.print_line('buf: ');
  DBE_output.print_line(buf);
END;
/
-- Expected result:
buf:
root
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.GETNAMEDITEM

  Returns the node specified by name. The prototype of the DBE_XMLDOM.GETNAMEDITEM function is as follows:

```
DBE_XMLDOM.GETNAMEDITEM(
  nnm IN DOMNAMEDNODEMAP,
  name IN VARCHAR2)
RETURN DOMNODE;
```

  Returns the node specified by name and namespace. The prototype of the DBE_XMLDOM.GETNAMEDITEM function is as follows:

```
DBE_XMLDOM.GETNAMEDITEM(
  nnm IN DOMNAMEDNODEMAP,
  name IN VARCHAR2,
  ns IN VARCHAR2)
RETURN DOMNODE;
```

  Table 10-441 DBE_XMLDOM.GETNAMEDITEM parameters

Parameter	Description
nnm	Specified DOMNamedNodeMap object
name	Name of the element to be retrieved
ns	Namespace

  NOTE

  1. The values of **name** and **nnm** can be null, but they are required arguments.
  2. The default maximum length of **name** and **ns** is 32767. If the length exceeds 32767, an error is reported.
  3. The values of **name** and **ns** can be of the int type and contain more than 127 bits.

  Example:

```
-- 1. Return the node specified by name.
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  elem DBE_XMLDOM.DOMElement;
  map DBE_XMLDOM.DOMNAMEDNODEMAP;
  node DBE_XMLDOM.DOMNODE;
```



```
  node2 DBE_XMLDOM.DOMNODE;
  buf varchar2(1000);
  buf2 varchar2(1000);
BEGIN
  doc := dbe_xmldom.newdomdocument('<bookstore category="web" cover="paperback">
     <book category="cooking"><title lang="en">Everyday Italian</title>
     <author>Giada De Laurentiis</author><year>2005</year>
     <price>30.00</price></book></bookstore>');
  elem := DBE_XMLDOM.GETDOCUMENTELEMENT(doc);
  node := DBE_XMLDOM.MAKENODE(elem);
  map := DBE_XMLDOM.GETATTRIBUTES(node);
  node2:= DBE_XMLDOM.GETNAMEDITEM(map,'category');
  DBE_XMLDOM.writeToBuffer(node2, buf2);
  dbe_output.print_line(buf2);
END;
/
-- Expected result:
web
ANONYMOUS BLOCK EXECUTE

-- 2. Return the node specified by name and namespace.
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  root DBE_XMLDOM.DOMElement;
  elem DBE_XMLDOM.DOMElement;
  map DBE_XMLDOM.DOMNAMEDNODEMAP;
  node DBE_XMLDOM.DOMNODE;
  buf varchar2(1000);
  buf2 varchar2(1000);
BEGIN
  doc := dbe_xmldom.newdomdocument('<h:table xmlns:h="http://www.w3.org/TR/html4/">
      <h:tr h:id="10"><h:td >Apples</h:td>
      <h:td>Bananas</h:td></h:tr></h:table>');
  root := DBE_XMLDOM.getDocumentElement(doc);
  node := DBE_XMLDOM.MAKENODE(root);
  node := dbe_xmldom.GETFIRSTCHILD(node);
  map := DBE_XMLDOM.GETATTRIBUTES(node);
  node := DBE_XMLDOM.GETNAMEDITEM(map,'id','http://www.w3.org/TR/html4/');
  DBE_XMLDOM.writeToBuffer(node, buf2);
  dbe_output.print_line(buf2);
END;
/
-- Expected result:
10
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.GETNEXTSIBLING

  Returns the next node. The prototype of the DBE_XMLDOM.GETNEXTSIBLING function is as follows:

```
DBE_XMLDOM.GETNEXTSIBLING(
  n  IN  DOMNODE)
RETURN DOMNODE;
```

Table 10-442 DBE_XMLDOM.GETNEXTSIBLING parameters

Parameter	Description
n	Specified DOMNode node

Example:

```
-- Obtain the first subnode after the DOC tree is converted into the DOMNode type, and obtain the name of the first subnode of the obtained first DOMNode node. Then, obtain the name of the next node through DBE_XMLDOM.GETNEXTSIBLING.
```



```
DECLARE
    doc dbe_xmldom.domdocument;
    doc_node dbe_xmldom.domnode;
    root_node dbe_xmldom.domnode;
    inside_node dbe_xmldom.domnode;
    node_name varchar2(1000);
    node_type integer;
BEGIN
   doc := dbe_xmldom.newdomdocument('<computer size="ITX">
        <cpu>Ryzen 9 3950X</cpu>
        <ram>32GBx2 DDR4 3200MHz</ram>
        <motherboard>X570i</motherboard>
    </computer>');
    doc_node := DBE_XMLDOM.MAKENODE(doc);
    root_node := DBE_XMLDOM.GETFIRSTCHILD(doc_node);
    node_name := DBE_XMLDOM.GETNODENAME(root_node);
    node_type := DBE_XMLDOM.GETNODETYPE(root_node);
    dbe_output.print_line(node_name);
    dbe_output.print_line(node_type);
    inside_node := DBE_XMLDOM.GETFIRSTCHILD(root_node);
    node_name := DBE_XMLDOM.GETNODENAME(inside_node);
    dbe_output.print_line(node_name);
    inside_node := DBE_XMLDOM.GETNEXTSIBLING(inside_node);
    node_name := DBE_XMLDOM.GETNODENAME(inside_node);
    dbe_output.print_line(node_name);
END;
/
-- Expected result:
computer
1
cpu
ram
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.GETNODENAME

  Returns the name of a node. The prototype of the DBE_XMLDOM.GETNODENAME function is as follows:

```
DBE_XMLDOM.GETNODENAME(
   n  IN  DOMNODE)
RETURN  VARCHAR2;
```

Table 10-443 DBE_XMLDOM.GETNODENAME parameters

Parameter	Description
n	Specified DOMNode node

Example:

```
-- Obtain the name of the specified DOMNode node from the DOC tree.
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  root DBE_XMLDOM.DOMElement;
  root_node DBE_XMLDOM.DOMNode;
  inside_node DBE_XMLDOM.DOMNode;
  buf VARCHAR2(1000);
BEGIN
   doc := dbe_xmldom.newdomdocument('<bookstore category="web" cover="paperback">
       <book category="cooking"><title lang="en">Everyday Italian</title>
       <author>Giada De Laurentiis</author><year>2005</year>
       <price>30.00</price></book></bookstore>');
  root := DBE_XMLDOM.getDocumentElement(doc);
  root_node := DBE_XMLDOM.MAKENODE(root);
  inside_node := DBE_XMLDOM.GETFIRSTCHILD(root_node);
```



```
  buf := DBE_XMLDOM.GETNODENAME(inside_node);
  dbe_output.print_line(buf);
END;
/
-- Expected result:
book
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.GETNODETYPE

  Returns the type of a node. The prototype of the DBE_XMLDOM.GETNODETYPE function is as follows:

```
DBE_XMLDOM.GETNODETYPE(
  n  IN  DOMNODE)
RETURN  NUMBER;
```

Table 10-444 DBE_XMLDOM.GETNODETYPE parameters

Parameter	Description
n	Specified DOMNode node

Example:

```
-- Obtain the type of the specified DOMNode node from the DOC tree.
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  doc_node DBE_XMLDOM.DOMNode;
  num number;
  buf varchar2(1000);
BEGIN
  doc := dbe_xmldom.newdomdocument('<bookstore category="web" cover="paperback">
     <book category="cooking"><title lang="en">Everyday Italian</title>
     <author>Giada De Laurentiis</author><year>2005</year>
     <price>30.00</price></book></bookstore>');
  doc_node := DBE_XMLDOM.makeNode(doc);
  num := DBE_XMLDOM.GETNODETYPE(doc_node);
  dbe_output.print_line(num);
  buf := DBE_XMLDOM.GETNODENAME(doc_node);
  dbe_output.print_line(buf);
END;
/
-- Expected result:
9
#document
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.GETNODEVALUE

  Returns the value of a DOMNode node. The prototype of the DBE_XMLDOM.GETNODEVALUE function is as follows:

```
DBE_XMLDOM.GETNODEVALUE(
  n  IN  DOMNODE)
RETURN  VARCHAR2;
```

Table 10-445 DBE_XMLDOM.GETNODEVALUE parameters

Parameter	Description
n	Specified DOMNode object

Example:

```
-- Convert a DOMText node to a DOMNode node and obtain the value of the node.
DECLARE
  buf VARCHAR2(1000);
  doc DBE_XMLDOM.DOMDocument;
  text DBE_XMLDOM.DOMText;
  elem2 DBE_XMLDOM.DOMElement;
  node DBE_XMLDOM.DOMNode;
begin
  doc := DBE_XMLDOM.NEWDOMDOCUMENT();
  text := DBE_XMLDOM.createTextNode(doc, 'aaa');
  DBE_XMLDOM.SETNODEVALUE(DBE_XMLDOM.makeNode(text), 'ccc');
  buf := DBE_XMLDOM.GETNODEVALUE(DBE_XMLDOM.makeNode(text));
  DBE_OUTPUT.print_line(buf);
end;
/
-- Expected result:
ccc
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.GETPARENTNODE

  Returns the parent node of the given DOMNode node. The prototype of the DBE_XMLDOM.GETPARENTNODE function is as follows:

```
DBE_XMLDOM.GETPARENTNODE(
   n  IN  DOMNODE)
RETURN  DOMNODE;
```

Table 10-446 DBE_XMLDOM.GETPARENTNODE parameters

Parameter	Description
n	Specified DOMNode object

Example:

```
-- Add a node to the DOC tree and obtain the name of its parent node.
DECLARE
   doc DBE_XMLDOM.DOMDocument;
   doc1 DBE_XMLDOM.DOMDocument;
   root DBE_XMLDOM.DOMElement;
   child1 DBE_XMLDOM.DOMElement;
   child2 DBE_XMLDOM.DOMElement;
   attr DBE_XMLDOM.DOMAttr;
   text DBE_XMLDOM.DOMTEXT;
   node DBE_XMLDOM.DOMNode;
   parent DBE_XMLDOM.DOMNode;
   buf varchar2(1000);
BEGIN
   doc := DBE_XMLDOM.newDOMDocument();
   root := DBE_XMLDOM.createElement(doc, 'root');
   node := DBE_XMLDOM.appendChild(DBE_xmldom.makeNode(doc),DBE_xmldom.makeNode(root));
   child1 := DBE_XMLDOM.createElement(doc, 'child1');
   node := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(root),
DBE_XMLDOM.makeNode(child1));
   child2 := DBE_XMLDOM.createElement(doc, 'child2');
   node := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(child1),
DBE_XMLDOM.makeNode(child2));
   parent := DBE_XMLDOM.GETPARENTNODE(DBE_XMLDOM.makeNode(child2));
   buf := DBE_XMLDOM.GETNODENAME(parent);
   DBE_OUTPUT.print_line(buf);
END;
/
-- Expected result:
```



```
child1
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.GETTAGNAME

  Returns the tag name of the specified DOMElement node. The prototype of the DBE_XMLDOM.GETTAGNAME function is as follows:

```
DBE_XMLDOM.GETTAGNAME(
  elem  IN  DOMELEMENT)
RETURN  VARCHAR2;
```

Table 10-447 DBE_XMLDOM.GETTAGNAME parameters

Parameter	Description
elem	Specified DOMElement node

Example:

```
-- Obtain the tag name of a DOMElement node.
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  buffer varchar2(1010);
  value  varchar(1000);
BEGIN
  doc := dbe_xmldom.newDOMDocument();
  elem := DBE_XMLDOM.CREATEELEMENT(DBE_XMLDOM.NEWDOMDOCUMENT(), 'root');
  value := DBE_XMLDOM.gettagname(elem);
  dbe_output.print_line('value: ');
  dbe_output.print_line(value);
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
END;
/
-- Expected result:
value:
root
buffer:
<?xml version="1.0" encoding="UTF-8"?>

ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.HASCHILDNODES

  Checks whether the DOMNode object has any subnode. The prototype of the DBE_XMLDOM.HASCHILDNODES function is as follows:

```
DBE_XMLDOM.HASCHILDNODES(
  n  IN  DOMNODE)
RETURN  BOOLEAN;
```

Table 10-448 DBE_XMLDOM.HASCHILDNODES parameters

Parameter	Description
n	Specified DOMNode object

Example:


```
-- Create a node named child1, mount it to the DOC tree, and add a node to child1. Then, check
whether the child1 node has any subnode.
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  doc1 DBE_XMLDOM.DOMDocument;
  root DBE_XMLDOM.DOMElement;
  child1 DBE_XMLDOM.DOMElement;
  child2 DBE_XMLDOM.DOMElement;
  attr DBE_XMLDOM.DOMAttr;
  text DBE_XMLDOM.DOMTEXT;
  node DBE_XMLDOM.DOMNode;
  buf varchar2(1000);
BEGIN
  doc := DBE_XMLDOM.newDOMDocument();
  root := DBE_XMLDOM.createElement(doc, 'root');
  node := DBE_XMLDOM.appendChild(DBE_xmldom.makeNode(doc),DBE_xmldom.makeNode(root));
  child1 := DBE_XMLDOM.createElement(doc, 'child1');
  node := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(root),
DBE_XMLDOM.makeNode(child1));
  child2 := DBE_XMLDOM.createElement(doc, 'child2');
  node := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(child1),
DBE_XMLDOM.makeNode(child2));
  IF DBE_XMLDOM.HASCHILDNODES(DBE_XMLDOM.makeNode(child1)) THEN
    DBE_OUTPUT.print_line('HAS CHILD NODES');
  ELSE
    DBE_OUTPUT.print_line('NOT HAS CHILD NODES ');
  END IF;
END;
/
-- Expected result:
HAS CHILD NODES
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.IMPORTNODE

  Copies a node to another node and mounts the copied node to a specified document. If the type of the copied node does not belong to the 12 types specified by constants of XML DOM, an exception indicating that the type is not supported is thrown. The prototype of the DBE_XMLDOM.IMPORTNODE function is as follows:

```
DBE_XMLDOM.IMPORTNODE(
  doc IN DOMDOCUMENT,
  importedNode IN DOMNODE,
  deep IN BOOLEAN)
RETURN DOMNODE;
```

Table 10-449 DBE_XMLDOM.IMPORTNODE parameters

Parameter	Description
doc	Document to which the node is mounted
importedNode	Node to be imported
deep	Specifies whether to perform recursive import. ● If the value is **TRUE**, the node and all its subnodes are imported. ● If the value is **FALSE**, the node itself is imported.

Example:

```
-- Obtain the root2_node node in the DOC2 tree, copy it, and mount it to the DOC tree.
DECLARE
    doc dbe_xmldom.domdocument;
    doc2 dbe_xmldom.domdocument;
    doc_node dbe_xmldom.domnode;
    doc2_node dbe_xmldom.domnode;
     root_node dbe_xmldom.domnode;
    root2_node dbe_xmldom.domnode;
    import_node dbe_xmldom.domnode;
    result_node dbe_xmldom.domnode;
    buffer varchar2(1010);
BEGIN
    doc := dbe_xmldom.newdomdocument('<bookstore category="web" cover="paperback">
       <book category="cooking"><title lang="en">Everyday Italian</title>
       <author>Giada De Laurentiis</author><year>2005</year>
       <price>30.00</price></book></bookstore>');
    doc2 := dbe_xmldom.newdomdocument('<case>LIANLI TU150</case>');
    doc_node := DBE_XMLDOM.MAKENODE(doc);
    doc2_node := DBE_XMLDOM.MAKENODE(doc2);
    root_node := DBE_XMLDOM.GETFIRSTCHILD(doc_node);
    root2_node := DBE_XMLDOM.GETFIRSTCHILD(doc2_node);
    DBE_XMLDOM.WRITETOBUFFER(doc, buffer);
    dbe_output.print_line(buffer);
    import_node := DBE_XMLDOM.IMPORTNODE(doc, root2_node, TRUE);
    result_node := DBE_XMLDOM.APPENDCHILD(root_node, import_node);
    DBE_XMLDOM.WRITETOBUFFER(doc, buffer);
    dbe_output.print_line(buffer);
END;
/
-- Expected result:
<?xml version="1.0" encoding="UTF-8"?>
<bookstore category="web" cover="paperback">
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
</bookstore>

<?xml version="1.0" encoding="UTF-8"?>
<bookstore category="web" cover="paperback">
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <case>LIANLI TU150</case>
</bookstore>

ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.ISNULL

  Checks whether the given DOMAttr node is null. If yes, **TRUE** is returned. Otherwise, **FALSE** is returned. The prototype of the DBE_XMLDOM.ISNULL function is as follows:

  ```
  DBE_XMLDOM.ISNULL(
    a       IN     DOMATTR)
  RETURN BOOLEAN;
  ```

  Checks whether the given DOMDocument node is null. If yes, **TRUE** is returned. Otherwise, **FALSE** is returned. The prototype of the DBE_XMLDOM.ISNULL function is as follows:


```
DBE_XMLDOM.ISNULL(
  doc      IN     DOMDOCUMENT)
RETURN BOOLEAN;
```

Checks whether the given DOMElement node is null. If yes, **TRUE** is returned. Otherwise, **FALSE** is returned. The prototype of the DBE_XMLDOM.ISNULL function is as follows:

```
DBE_XMLDOM.ISNULL(
  elem     IN     DOMELEMENT)
RETURN BOOLEAN;
```

Checks whether the given DOMNamedNodeMap node is null. If yes, **TRUE** is returned. Otherwise, **FALSE** is returned. The prototype of the DBE_XMLDOM.ISNULL function is as follows:

```
DBE_XMLDOM.ISNULL(
  nnm      IN     DOMNAMEDNODEMAP)
RETURN BOOLEAN;
```

Checks whether the given DOMNode node is null. If yes, **TRUE** is returned. Otherwise, **FALSE** is returned. The prototype of the DBE_XMLDOM.ISNULL function is as follows:

```
DBE_XMLDOM.ISNULL(
  n      IN     DOMNODE)
RETURN BOOLEAN;
```

Checks whether the given DOMNodeList node is null. If yes, **TRUE** is returned. Otherwise, **FALSE** is returned. The prototype of the DBE_XMLDOM.ISNULL function is as follows:

```
DBE_XMLDOM.ISNULL(
  nl       IN     DOMNODELIST)
RETURN BOOLEAN;
```

Checks whether the given DOMText node is null. If yes, **TRUE** is returned. Otherwise, **FALSE** is returned. The prototype of the DBE_XMLDOM.ISNULL function is as follows:

```
DBE_XMLDOM.ISNULL(
  t      IN     DOMTEXT)
RETURN BOOLEAN;
```

Table 10-450 DBE_XMLDOM.ISNULL parameters

Parameter	Description
a	Specified DOMAttr node
doc	Specified DOMDocument node
elem	Specified DOMElement node
nnm	Specified DOMNamedNodeMap node
n	Specified DOMNode node
nl	Specified DOMNodeList node
t	Specified DOMText node

NOTE

Due to the implementation difference of DBE_XMLDOM.FREEDOCUMENT, an error is reported when the DBE_XMLDOM.ISNULL API calls the freed DOMDocument node.

Example:

```
-- 1. Use createAttribute to create a DOMAttr node and check whether the node is empty.
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  attr DBE_XMLDOM.DOMATTR;
  buf VARCHAR2(1000);
BEGIN
  doc := DBE_xmldom.newdomdocument('<?xml version="1.0"?>
    <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)>
    <!ATTLIST note color CDATA #REQUIRED>
    <!ELEMENT to (#PCDATA)>
    <!ELEMENT from (#PCDATA)>
    <!ELEMENT heading (#PCDATA)>
    <!ELEMENT body (#PCDATA)>]>
    <note color="red"><to>Chinese</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don''t forget me this weekend!</body>
    </note>');
  attr := DBE_XMLDOM.CREATEATTRIBUTE (doc, 'length');
  if DBE_XMLDOM.ISNULL(attr) then
    DBE_OUTPUT.print_line('null');
  else
    DBE_OUTPUT.print_line('not null');
  end if;
END;
/
-- Expected result:
not null
ANONYMOUS BLOCK EXECUTE

-- 2. Declare (but not initialize) a DOMElement node and check whether the node is empty.
DECLARE
  docelem   DBE_XMLDOM.DOMElement;
BEGIN
  if DBE_XMLDOM.ISNULL(docelem) then
    DBE_OUTPUT.print_line('null');
  else
    DBE_OUTPUT.print_line('not null');
  end if;
END;
/
-- Expected result:
null
ANONYMOUS BLOCK EXECUTE

-- 3. Use newDomdocument to construct a good DOMDocument node and check whether the node is
empty.
Declare
 doc dbe_xmldom.domdocument;
BEGIN
 doc := DBE_xmldom.newdomdocument('<?xml version="1.0"?>
    <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)>
    <!ATTLIST note color CDATA #REQUIRED>
    <!ELEMENT to (#PCDATA)>
    <!ELEMENT from (#PCDATA)>
    <!ELEMENT heading (#PCDATA)>
    <!ELEMENT body (#PCDATA)>]>
    <note color="red"><to>Chinese</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don''t forget me this weekend!</body>
    </note>');
 if DBE_XMLDOM.ISNULL(doc) then
    DBE_OUTPUT.print_line('null');
  else
    DBE_OUTPUT.print_line('not null');
  end if;
END;
```



```
/
-- Expected result:
not null
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.ITEM

  Returns the element corresponding to the index in a list based on the index. The prototype of the DBE_XMLDOM.ITEM function is as follows:

```
DBE_XMLDOM.ITEM(
  nl IN DOMNODELIST,
  index IN NUMBER)
RETURN DOMNODE;
```

  Returns the element corresponding to the index in a map based on the index. The prototype of the DBE_XMLDOM.ITEM function is as follows:

```
DBE_XMLDOM.ITEM(
  nnm IN DOMNAMEDNODEMAP,
  index IN NUMBER)
RETURN DOMNODE;
```

Table 10-451 DBE_XMLDOM.ITEM parameters

Parameter	Description
nl	Specified DOMNodeList object
nnm	Specified DOMNamedNodeMap object
index	Index of the element to be retrieved

NOTE

For improper input parameters such as Boolean and CLOB, the item function of the map type points to the value of the first index by default.

Example:

```
-- 1. Return the element corresponding to the index in a map based on the index.
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  elem DBE_XMLDOM.DOMElement;
  map DBE_XMLDOM.DOMNAMEDNODEMAP;
  node DBE_XMLDOM.DOMNODE;
  node2 DBE_XMLDOM.DOMNODE;
  buf varchar2(1000);
BEGIN
  doc := dbe_xmldom.newdomdocument('<bookstore category="web" cover="paperback"><book
category="cooking">
      <title lang="en">Everyday Italian</title><author>Giada De Laurentiis</author>
      <year>2005</year><price>30.00</price></book></bookstore>');
  elem := DBE_XMLDOM.GETDOCUMENTELEMENT(doc);
  node := DBE_XMLDOM.MAKENODE(elem);
  map := DBE_XMLDOM.GETATTRIBUTES(DBE_XMLDOM.getFirstChild(node));
  node2:= DBE_XMLDOM.item(map,0);
  DBE_XMLDOM.writeToBuffer(node2, buf);
  dbe_output.print_line(buf);
  dbe_xmldom.freedocument(doc);
  RAISE NOTICE '%', buf;
END;
/
-- Expected result:
cooking
```



```
NOTICE:  cooking
ANONYMOUS BLOCK EXECUTE

-- 2. Return the element corresponding to the index in a list based on the index.
DECLARE
  doc dbe_xmldom.domdocument;
  node dbe_xmldom.domnode;
  node1 dbe_xmldom.domnode;
  nodelist DBE_XMLDOM.DOMNODELIST;
  len INTEGER;
  buffer1 varchar2(1010);
BEGIN
  doc := dbe_xmldom.newdomdocument('<bookstore category="web" cover="paperback"><book
category="cooking">
      <title lang="en">Everyday Italian</title><author>Giada De Laurentiis</author>
      <year>2005</year><price>30.00</price></book></bookstore>');
  node := dbe_xmldom.makenode(doc);
  node := dbe_xmldom.GETFIRSTCHILD(node);
  node := dbe_xmldom.GETFIRSTCHILD(node);
  nodelist := DBE_XMLDOM.GETCHILDNODES(node);
  len := DBE_XMLDOM.GETLENGTH(nodelist);
  RAISE NOTICE 'len :  %', len;
  node1 := DBE_XMLDOM.ITEM(nodelist, 0);
  IF DBE_XMLDOM.ISNULL(node1) THEN
       dbe_output.print_line('IS NULL');
  ELSE
       dbe_output.print_line('NOT NULL');
  END IF;
  dbe_xmldom.writetobuffer(node1, buffer1);
  dbe_output.print_line('buffer1: ');
  dbe_output.print_line(buffer1);
END;
/
-- Expected result:
NOTICE:  len :  4
NOT NULL
buffer1:
<title lang="en">Everyday Italian</title>
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.MAKEELEMENT

  Returns the DOMElement object after conversion. The prototype of the DBE_XMLDOM.MAKEELEMENT function is as follows:

```
DBE_XMLDOM.MAKEELEMENT(
  n IN DOMNODE)
RETURN DOMELEMENT;
```

Table 10-452 DBE_XMLDOM.MAKEELEMENT parameters

Parameter	Description
n	Specified DOMNode object

Example:

```
-- Forcibly convert the DOMNode node converted from the DOMElement type back to the
DOMElement type.
DECLARE
  buf VARCHAR2(1000);
  doc DBE_XMLDOM.DOMDocument;
  elem DBE_XMLDOM.DOMElement;
  elem2 DBE_XMLDOM.DOMElement;
  node DBE_XMLDOM.DOMNode;
BEGIN
```



```
  doc := DBE_XMLDOM.NEWDOMDOCUMENT();
  elem := DBE_XMLDOM.createElement(doc, 'aaa');
  node := DBE_XMLDOM.makeNode(elem);
  elem2 := DBE_XMLDOM.makeElement(node);
  buf := DBE_XMLDOM.GETNODENAME(DBE_XMLDOM.makeNode(elem2));
  DBE_OUTPUT.print_line(buf);
END;
/
-- Expected result:
aaa
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.MAKENODE

  Forcibly converts a specified DOMAttr node to a DOMNode node and returns the DOMNode node. The prototype of the DBE_XMLDOM.MAKENODE function is as follows:

  ```
  DBE_XMLDOM.MAKENODE(
    a        IN     DOMATTR)
  RETURN DOMNODE;
  ```

  Forcibly converts a specified DOMDocument node to a DOMNode node and returns the DOMNode node. The prototype of the DBE_XMLDOM.MAKENODE function is as follows:

  ```
  DBE_XMLDOM.MAKENODE(
    doc      IN     DOMDOCUMENT)
  RETURN DOMNODE;
  ```

  Forcibly converts a specified DOMElement node to a DOMNode node and returns the DOMNode node. The prototype of the DBE_XMLDOM.MAKENODE function is as follows:

  ```
  DBE_XMLDOM.MAKENODE(
    elem       IN     DOMELEMENT)
  RETURN DOMNODE;
  ```

  Forcibly converts a specified DOMText node to a DOMNode node and returns the DOMNode node. The prototype of the DBE_XMLDOM.MAKENODE function is as follows:

  ```
  DBE_XMLDOM.MAKENODE(
    t       IN     DOMTEXT)
  RETURN DOMNODE;
  ```

  Table 10-453 DBE_XMLDOM.MAKENODE parameters

Parameter	Description
a	Specified DOMAttr node
doc	Specified DOMDocument node
elem	Specified DOMElement node
t	Specified DOMText node

NOTE

Due to syntax restrictions, when DBE_XMLDOM.MAKENODE is used as the return value of a function, it cannot be directly implemented by running the following command:

```
return DBE_XMLDOM.MAKENODE(doc);
```

You are advised to run the following command:

```
tmp_node := DBE_XMLDOM.MAKENODE(doc );
return tmp_node;
```

Example:

```
-- 1. Use createattr to generate ATTR and convert it to a node.
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  attr DBE_XMLDOM.DOMATTR;
  dom_node DBE_XMLDOM.DOMNode;
  buf VARCHAR2(1000);
BEGIN
  doc := DBE_xmldom.newdomdocument('<?xml version="1.0"?>
      <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)>
      <!ELEMENT to (#PCDATA)>
      <!ELEMENT from (#PCDATA)>
      <!ELEMENT heading (#PCDATA)>
      <!ELEMENT body (#PCDATA)>]>
      <note><to>Chinese</to>
      <from>Jani</from>
      <heading>Reminder</heading>
      <body>Don''t forget me this weekend!</body>
      </note>');
   attr := DBE_XMLDOM.CREATEATTRIBUTE (doc, 'length');
   dom_node := DBE_XMLDOM.makeNode(attr);
   buf := DBE_XMLDOM.getNodeName(dom_node);
   DBE_OUTPUT.print_line(buf);
END;
/
-- Expected result:
length
ANONYMOUS BLOCK EXECUTE

-- 2. Use the getdocumentelement function to generate an elem node and then perform makenode.
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  root DBE_XMLDOM.DOMElement;
  attr DBE_XMLDOM.DOMATTR;
  node DBE_XMLDOM.DOMNODE;
  buf VARCHAR2(1000);
BEGIN
  doc := DBE_xmldom.newdomdocument('<?xml version="1.0"?>
      <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)>
      <!ATTLIST note color CDATA #REQUIRED>
      <!ELEMENT to (#PCDATA)>
      <!ELEMENT from (#PCDATA)>
      <!ELEMENT heading (#PCDATA)>
      <!ELEMENT body (#PCDATA)>]>
      <note color="red"><to>Chinese</to>
      <from>Jani</from>
      <heading>Reminder</heading>
      <body>Don''t forget me this weekend!</body>
      </note>');
  root := DBE_XMLDOM.getDocumentElement(doc);
  node := DBE_XMLDOM.makenode(root);
  DBE_OUTPUT.print_line(DBE_XMLDOM.GETNODENAME(node));
END;
/
-- Expected result:
note
ANONYMOUS BLOCK EXECUTE
```



```
-- 3. Use newdomdocument to create a parameter of the DOMDocument type. The parameter is not
empty and is used as the input parameter of MAKENODE.
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  buf VARCHAR2(1000);
  dom_node DBE_XMLDOM.DOMNODE;
BEGIN
  doc := DBE_xmldom.newdomdocument('<?xml version="1.0"?>
            <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)>
            <!ELEMENT to (#PCDATA)>
            <!ELEMENT from (#PCDATA)>
            <!ELEMENT heading (#PCDATA)>
            <!ELEMENT body (#PCDATA)>]>
      <note><to>Chinese</to>
            <from>Jani</from>
            <heading>Reminder</heading>
            <body>Don''t forget me this weekend!</body>
            </note>');
  DBE_OUTPUT.print_line('doc.id: ');
  DBE_OUTPUT.print_line(doc.id);
  dom_node := DBE_XMLDOM.makeNode(doc);
  DBE_OUTPUT.print_line('dom_node.id: ');
  DBE_OUTPUT.print_line(dom_node.id);
  buf := DBE_XMLDOM.GETNODENAME(dom_node);
  DBE_OUTPUT.print_line(buf);
END;
/
-- Expected result:
doc.id:
19000000000000001B00000001
dom_node.id:
19000000010000001B00000001
#document
ANONYMOUS BLOCK EXECUTE

-- 4. Declare (but not initialize) a DOMText variable, and use it as the input parameter of MAKENODE.
DECLARE
  text DBE_XMLDOM.DOMTEXT;
  buf VARCHAR2(1000);
  dom_node DBE_XMLDOM.DOMNODE;
BEGIN
  dom_node := DBE_XMLDOM.makeNode(text);
  buf := DBE_XMLDOM.GETNODENAME(dom_node);
  DBE_OUTPUT.print_line(buf);
END;
/
-- Expected result:
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.NEWDOMDOCUMENT

  Returns a new DOMDocument object. The prototype of the DBE_XMLDOM.NEWDOMDOCUMENT function is as follows:

  ```
  DBE_XMLDOM.NEWDOMDOCUMENT
  RETURN DOMDOCUMENT;
  ```

  Returns a new DOMDocument instance object created from the specified XMLType type. The prototype of the DBE_XMLDOM.NEWDOMDOCUMENT function is as follows:

  ```
  DBE_XMLDOM.NEWDOMDOCUMENT(
    xmldoc    IN SYS.XMLTYPE)
  RETURN DOMDOCUMENT;
  ```

  Returns a new DOMDocument instance object created from the specified CLOB type. The prototype of the DBE_XMLDOM.NEWDOMDOCUMENT function is as follows:


```
DBE_XMLDOM.NEWDOMDOCUMENT(
  cl      IN    CLOB)
RETURN DOMDOCUMENT;
```

Table 10-454 DBE_XMLDOM.NEWDOMDOCUMENT parameters

Parameter	Description
xmldoc	Specified XMLType type
cl	Specified CLOB type

NOTE

- The size of the input parameter must be less than 2 GB.
- Currently, external DTD parsing is not supported.
- The document created by NEWDOMDOCUMENT uses the UTF-8 character set by default.
- Each document parsed from the same XMLType instance is independent, and the modification of the document does not affect the XMLType.
- For details about the differences between our database and database A, see **DBE_XMLPARSER.PARSECLOB**.

Example:

```
-- 1. Return a new DOMDocument object.
DECLARE
  doc dbe_xmldom.domdocument;
  buffer varchar2(1010);
BEGIN
  doc := dbe_xmldom.newdomdocument();
  dbe_xmldom.setdoctype(doc, 'note', 'sysid', 'pubid');
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
  dbe_xmldom.freedocument(doc);
END;
/
-- Expected result:
buffer:
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note PUBLIC "pubid" "sysid">

ANONYMOUS BLOCK EXECUTE

-- 2. Return a new DOMDocument instance object created from the specified CLOB type.
DECLARE
  doc dbe_xmldom.domdocument;
  buffer varchar2(1010);
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
      <note><to>test</to><from>Jani</from><heading>Reminder</heading>
      <body>Don''t forget me this weekend!</body></note>');
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
  dbe_xmldom.freedocument(doc);
END;
/
-- Expected result:
buffer:
<?xml version="1.0" encoding="UTF-8"?>
<note>
```



```
 <to>test</to>
 <from>Jani</from>
 <heading>Reminder</heading>
 <body>Don't forget me this weekend!</body>
</note>

ANONYMOUS BLOCK EXECUTE

-- 3. Return a new DOMDocument instance object created from the specified XMLType type.
DECLARE
doc dbe_xmldom.domdocument;
  xt xmltype;
  buffer varchar2(1010);
BEGIN
  xt := xmltype('<h:data xmlns:h="http://www.w3.org/TR/html4/">
    <h:da1 len="10">test namespace</h:da1>
    <h:da1>bbbbbbbbbb</h:da1>
    </h:data>');
  doc := dbe_xmldom.newdomdocument(xt);
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
  dbe_xmldom.freedocument(doc);
END;
/
-- Expected result:
buffer:
<?xml version="1.0" encoding="UTF-8"?>
<h:data xmlns:h="http://www.w3.org/TR/html4/">
 <h:da1 len="10">test namespace</h:da1>
 <h:da1>bbbbbbbbbb</h:da1>
</h:data>

ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.SETATTRIBUTE

  Sets the value of the DOMElement attribute by name. The prototype of the DBE_XMLDOM.SETATTRIBUTE function is as follows:

  ```
  DBE_XMLDOM.SETATTRIBUTE(
    elem    IN  DOMELEMENT,
    name    IN  VARCHAR2,
    value   IN  VARCHAR2);
  ```

  Sets the attribute values of a DOMElement object by name and namespace URI. The prototype of the DBE_XMLDOM.SETATTRIBUTE function is as follows:

  ```
  DBE_XMLDOM.SETATTRIBUTE(
    elem    IN  DOMELEMENT,
    name    IN  VARCHAR2,
    value   IN  VARCHAR2,
    ns      IN  VARCHAR2);
  ```

  Table 10-455 DBE_XMLDOM.SETATTRIBUTE parameters

Parameter	Description
elem	Specified DOMElement node
name	Attribute name
value	Attribute value
ns	Namespace

> **NOTE**
>
> Multiple attributes can be added through the DBE_XMLDOM.SETATTRIBUTE API. The attribute name cannot be null, and attributes with the same name cannot exist in the same DOMElement node. If you want to add attributes with the same name, you should explicitly set a namespace for each attribute with the same name, but you are advised not to perform such operations. If an attribute exists in a namespace, the specified namespace must be displayed when you modify the attribute. Otherwise, the attribute with the same name is added.

Example:

```
-- 1. Set the value of the DOMElement attribute by name.
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  docnode DBE_XMLDOM.DOMNode;
  buffer varchar2(1010);
  value  varchar(1000);
BEGIN
  doc := dbe_xmldom.newDOMDocument();
  elem := DBE_XMLDOM.CREATEELEMENT(doc, 'root');
  DBE_XMLDOM.setattribute(elem, 'len', '50cm');
  docnode := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(doc),
DBE_XMLDOM.makeNode(elem));
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
END;
/
-- Expected result:
buffer:
<?xml version="1.0" encoding="UTF-8"?>
<root len="50cm"/>

ANONYMOUS BLOCK EXECUTE

-- 2. Set the attribute values of a DOMElement object by name and namespace URI.
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  docnode DBE_XMLDOM.DOMNode;
  buffer varchar2(1010);
  value  varchar(1000);
begin
  doc := dbe_xmldom.newDOMDocument();
  elem := DBE_XMLDOM.CREATEELEMENT(doc, 'root');
  DBE_XMLDOM.setattribute(elem, 'len', '50cm', 'www.huawei.com');
  docnode := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(doc),
DBE_XMLDOM.makeNode(elem));
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
END;
/
-- Expected result:
buffer:
<?xml version="1.0" encoding="UTF-8"?>
<root len="50cm"/>

ANONYMOUS BLOCK EXECUTE

-- 3. Change the values of the DOMElement attributes by name.
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  docnode DBE_XMLDOM.DOMNode;
  buffer varchar2(1010);
  value  varchar(1000);
```



```
BEGIN
  doc := dbe_xmldom.newDOMDocument();
  elem := DBE_XMLDOM.CREATEELEMENT(doc, 'root');
  DBE_XMLDOM.setattribute(elem, 'len', '50cm');
  DBE_XMLDOM.setattribute(elem, 'len', '55cm');
  docnode := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(doc),
DBE_XMLDOM.makeNode(elem));
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
END;
/
-- Expected result:
buffer:
<?xml version="1.0" encoding="UTF-8"?>
<root len="55cm"/>

ANONYMOUS BLOCK EXECUTE

-- 4. Change the values of the DOMElement attributes by name and namespace URI.
DECLARE
  doc dbe_xmldom.domdocument;
  elem dbe_xmldom.domelement;
  docnode DBE_XMLDOM.DOMNode;
  buffer varchar2(1010);
  value  varchar(1000);
begin
  doc := dbe_xmldom.newDOMDocument();
  elem := DBE_XMLDOM.CREATEELEMENT(doc, 'root');
  DBE_XMLDOM.setattribute(elem, 'len', '50cm', 'www.huawei.com');
  DBE_XMLDOM.setattribute(elem, 'len', '55cm', 'www.huawei.com');
  docnode := DBE_XMLDOM.appendChild(DBE_XMLDOM.makeNode(doc),
DBE_XMLDOM.makeNode(elem));
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
END;
/
-- Expected result:
buffer:
<?xml version="1.0" encoding="UTF-8"?>
<root len="55cm"/>

ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.SETCHARSET

  Sets the character set for a DOMDocument object. The prototype of the DBE_XMLDOM.SETCHARSET function is as follows:

```
DBE_XMLDOM.SETCHARSET(
  doc       IN     DOMDocument,
  charset   IN     VARCHAR2);
```

Table 10-456 DBE_XMLDOM.SETCHARSET parameters

Parameter	Description
doc	Specified DOMDocument node
charset	Character set

NOTE

- The value of **charset** contains a maximum of 60 bytes.
- Currently, the following character sets are supported: UTF-8, UTF-16, UCS-4, UCS-2, ISO-8859-1, ISO-8859-2, ISO-8859-3, ISO-8859-4, ISO-8859-5, ISO-8859-6, ISO-8859-7, ISO-8859-8, ISO-8859-9, ISO-2022-JP, Shift_JIS, EUC-JP and ASCII. If you enter other character sets, an error is reported or garbled characters may be displayed.

Example:

```
-- Set the UTF-16 character set for the DOC tree and print the DOC tree to the buffer.
DECLARE
  doc dbe_xmldom.domdocument;
  buffer varchar2(1010);
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
     <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)><!ELEMENT to (#PCDATA)>
     <!ELEMENT from (#PCDATA)><!ELEMENT heading (#PCDATA)><!ELEMENT body (#PCDATA)>]>
     <note><to>test</to><from>Jani</from><heading>Reminder</heading>
     <body>Don''t forget me this weekend!</body></note>');
  dbe_xmldom.setcharset(doc, 'utf-16');
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
  dbe_xmldom.freedocument(doc);
END;
/
-- Expected result:
buffer:
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note [
<!ELEMENT note (to , from , heading , body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
  <to>test</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>

ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.SETDOCTYPE

  Sets the external DTD of a DOMDocument object. The prototype of the DBE_XMLDOM.SETDOCTYPE function is as follows:

```
DBE_XMLDOM.SETDOCTYPE(
  doc     IN   DOMDocument,
  name    IN   VARCHAR2,
  sysid   IN   VARCHAR2,
  pubid   IN   VARCHAR2);
```

Table 10-457 DBE_XMLDOM.SETDOCTYPE parameters

Parameter	Description
doc	Specified DOMDocument node
name	Name of the DOCType to be initialized

Parameter	Description
sysid	ID of the system whose DOCType needs to be initialized
pubid	Public ID of the DOCType to be initialized

NOTE

The total length of **name**, **sysid**, and **pubid** cannot exceed 32500 bytes.

Example:

```
-- After the initial system ID, public ID, and name are set for the external DTD of the DOMDocument, the DOC tree modified each time is output to the buffer.
DECLARE
  doc dbe_xmldom.domdocument;
  buffer varchar2(1010);
begin
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
     <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)><!ELEMENT to (#PCDATA)>
     <!ELEMENT from (#PCDATA)><!ELEMENT heading (#PCDATA)><!ELEMENT body (#PCDATA)>]>
     <note><to>test</to><from>Jani</from><heading>Reminder</heading>
     <body>Don''t forget me this weekend!</body></note>');
  dbe_xmldom.setdoctype(doc, 'note', 'sysid', 'pubid');
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
  dbe_output.print_line('-------------------------------------------------');
  dbe_xmldom.setdoctype(doc, 'n0te', NULL, '');
  dbe_xmldom.setdoctype(doc, 'n0t1e', NULL, '');
  dbe_xmldom.writetobuffer(doc, buffer);
  dbe_output.print_line('buffer: ');
  dbe_output.print_line(buffer);
  dbe_xmldom.freedocument(doc);
END;
/
-- Expected result:
buffer:
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note PUBLIC "pubid" "sysid" [
<!ELEMENT note (to , from , heading , body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
  <to>test</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>

-------------------------------------------------
buffer:
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE n0t1e [
<!ELEMENT note (to , from , heading , body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
```



```
 <to>test</to>
 <from>Jani</from>
 <heading>Reminder</heading>
 <body>Don't forget me this weekend!</body>
</note>

ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.SETNODEVALUE

  Sets the value of a node in the DOMNode object. The prototype of the DBE_XMLDOM.SETNODEVALUE function is as follows:

```
DBE_XMLDOM.SETNODEVALUE(
 n IN DOMNODE,
 nodeValue IN VARCHAR2);
```

Table 10-458 DBE_XMLDOM.SETNODEVALUE parameters

Parameter	Description
n	Specified DOMNode object
nodeValue	String to be set in the DOMNode object

NOTE

1. You can enter an empty string or null value for **nodeValue**, but the node value will not be changed.
2. Currently, **nodeValue** does not support the escape character '&'. If the character string contains the escape character, the node value will be cleared.
3. The default maximum length of **nodeValue** is restricted by the VARCHAR2 type and is 32767 bytes. If the length exceeds 32767 bytes, an exception is thrown.

Example:

```
-- After setting a node value different from the initial value for the DOMNode node that is converted from DOMText, obtain and output the node value.
DECLARE
  buf VARCHAR2(1000);
  doc DBE_XMLDOM.DOMDocument;
  text DBE_XMLDOM.DOMText;
  elem2 DBE_XMLDOM.DOMElement;
  node DBE_XMLDOM.DOMNode;
BEGIN
  doc := DBE_XMLDOM.NEWDOMDOCUMENT();
  text := DBE_XMLDOM.createTextNode(doc, 'aaa');
  DBE_XMLDOM.SETNODEVALUE(DBE_XMLDOM.makeNode(text), 'ccc');
  buf := DBE_XMLDOM.GETNODEVALUE(DBE_XMLDOM.makeNode(text));
  DBE_OUTPUT.print_line(buf);
END;
/
-- Expected result:
ccc
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.WRITETOBUFFER

  Writes an XML node to a specified buffer using the database character set. The prototype of the DBE_XMLDOM.WRITETOBUFFER function is as follows:


```
DBE_XMLDOM.WRITETOBUFFER(
  doc      IN     DOMDOCUMENT,
  buffer   INOUT  VARCHAR2);
```

Writes an XML document to a specified buffer using the database character set. The prototype of the DBE_XMLDOM.WRITETOBUFFER function is as follows:

```
DBE_XMLDOM.WRITETOBUFFER(
  n        IN     DOMNODE,
  buffer   INOUT  VARCHAR2);
```

Table 10-459 DBE_XMLDOM.WRITETOBUFFER parameters

Parameter	Description
doc	Specified DOMDocument node
buffer	Buffer for the write operation
n	Specified DOMNode node

NOTE

- The buffer to which the writetobuffer function writes is less than 1 GB.
- This function adds content such as indentation to format the output. The output document will contain the XML declaration version and encoding.
- By default, XML files are output in the UTF-8 character set.

Example:

```
-- 1. Enter a parameter of the DOMNode type.
DECLARE
   doc dbe_xmldom.domdocument;
   elem DBE_XMLDOM.DOMELEMENT;
   buf varchar2(1000);
BEGIN
    doc := dbe_xmldom.newdomdocument();
    elem := dbe_xmldom.createelement(doc,'elem');
    DBE_XMLDOM.WRITETOBUFFER(dbe_xmldom.makenode(elem), buf);
    DBE_OUTPUT.print_line(buf);
END;
/
-- Expected result:
<elem/>
ANONYMOUS BLOCK EXECUTE

-- 2. Enter a parameter of the DOMDocument type.
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  buf VARCHAR2(1000);
BEGIN
  doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
       <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)><!ELEMENT to (#PCDATA)>
       <!ELEMENT from (#PCDATA)><!ELEMENT heading (#PCDATA)><!ELEMENT body (#PCDATA)>]>
       <note><to>test</to><from>Jani</from><heading>Reminder</heading>
       <body>Don''t forget me this weekend!</body></note>');
  DBE_XMLDOM.WRITETOBUFFER(doc, buf);
  DBE_OUTPUT.print_line('doc: ');
  DBE_OUTPUT.print_line(buf);
  DBE_XMLDOM.FREEDOCUMENT(doc);
END;
/
-- Expected result:
doc:
```



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note [
<!ELEMENT note (to , from , heading , body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
  <to>test</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>

ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.WRITETOCLOB

  Writes an XML node to a specified CLOB using the database character set. The prototype of the DBE_XMLDOM.WRITETOCLOB function is as follows:

```
DBE_XMLDOM.WRITETOCLOB(
  doc     IN      DOMDOCUMENT,
  cl      INOUT  CLOB);
```

  Writes an XML node to a specified CLOB using the database character set. The prototype of the DBE_XMLDOM.WRITETOCLOB function is as follows:

```
DBE_XMLDOM.WRITETOCLOB(
  n       IN      DOMNODE,
  cl      INOUT  CLOB);
```

Table 10-460 DBE_XMLDOM.WRITETOCLOB parameters

Parameter	Description
doc	Specified DOMDocument node
cl	CLOB to be written
n	Specified DOMNode node

NOTE

- The **doc** parameter is an input parameter. The value of **writetoclob** is less than 2 GB.
- This function adds content such as indentation to format the output. The output document will contain the XML declaration version and encoding.
- By default, XML files are output in the UTF-8 character set.

Example:

```
-- 1. Enter a parameter of the DOMNode type.
DECLARE
  CL  CLOB;
  N   DBE_XMLDOM.DOMNODE;
BEGIN
  DBE_XMLDOM.WRITETOCLOB(N, CL);
  DBE_OUTPUT.PRINT_LINE(CL);
END;
/
-- Expected result:
ANONYMOUS BLOCK EXECUTE
```



```
-- 2. Enter a parameter of the DOMDocument type.
DECLARE
    doc dbe_xmldom.domdocument;
    mclob clob;
BEGIN
    doc := dbe_xmldom.newdomdocument('<?xml version="1.0"?>
       <!DOCTYPE note [<!ELEMENT note (to,from,heading,body)><!ELEMENT to (#PCDATA)>
       <!ELEMENT from (#PCDATA)><!ELEMENT heading (#PCDATA)><!ELEMENT body (#PCDATA)>]>
       <note><to>test</to><from>Jani</from><heading>Reminder</heading>
       <body>Don''t forget me this weekend!</body></note>');
    dbe_xmldom.writetoclob(doc, mclob);
    dbe_output.print_line('mclob: ');
    dbe_output.print_line(mclob);
    dbe_xmldom.freedocument(doc);
END;
/
-- Expected result:
mclob:
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note [
<!ELEMENT note (to , from , heading , body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
  <to>test</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>

ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.WRITETOFILE

  Writes an XML node to a specified file using the database character set. The prototype of the DBE_XMLDOM.WRITETOFILE function is as follows:

  ```
  DBE_XMLDOM.WRITETOCLOB(
    doc     IN      DOMDOCUMENT,
    fileName   IN      VARCHAR2);
  ```

  Writes an XML node to a specified file using the database character set. The prototype of the DBE_XMLDOM.WRITETOFILE function is as follows:

  ```
  DBE_XMLDOM.WRITETOCLOB(
    n       IN      DOMNODE,
    fileName   IN      VARCHAR2);
  ```

  Writes an XML document to a specified file using the specified character set. The prototype of the DBE_XMLDOM.WRITETOFILE function is as follows:

  ```
  DBE_XMLDOM.WRITETOCLOB(
    doc     IN      DOMDOCUMENT,
    fileName   IN      VARCHAR2,
    charset   IN   VARCHAR2);
  ```

  Writes an XML document to a specified file using the specified character set. The prototype of the DBE_XMLDOM.WRITETOFILE function is as follows:

  ```
  DBE_XMLDOM.WRITETOCLOB(
    n       IN      DOMNODE,
    fileName   IN      VARCHAR2,
    charset   IN   VARCHAR2);
  ```


Table 10-461 DBE_XMLDOM.WRITETOFILE parameters

Parameter	Description
doc	Specified DOMDocument node
fileName	File to be written
n	Specified DOMNode node
charset	Specified character set

NOTE

- The **doc** parameter is an input parameter. The value of **filename** can contain a maximum of 255 bytes, and the value of **charset** can contain a maximum of 60 bytes. For details about the character sets supported by **charset**, see the **DBE_XMLDOM.SETCHARSET** API.
- This function adds content such as indentation to format the output. The output document will contain the XML declaration version and encoding.
- If newdomdocument() is used to create a document without parameters, no error is reported when **charset** is not specified. The UTF-8 character set is used by default.
- The **filename** must be in the path created in **pg_directory**. The backslash (\) in the filename will be converted to a slash (/). Only one slash (/) is allowed. The file name must be in the **pg_directory_name/file_name.xml** format. The output file must be in the XML format.
- When the GUC parameter **safe_data_path** is enabled, you can only use an advanced package to read and write files in the file path specified by **safe_data_path**.
- Before creating a directory, ensure that the directory exists in the operating system and the user has the read and write permissions on the directory. For details about how to create a directory, see **CREATE DIRECTORY**.

Example:

```
-- Before creating a directory, ensure that the directory exists in the OS and the user has the read and write permissions on the directory.
create directory dir as '/tmp';
-- 1. Write an XML node to a specified file using the database character set.
DECLARE
  FPATH VARCHAR2(1000);
  DOC   DBE_XMLDOM.DOMDOCUMENT;
BEGIN
  DOC := DBE_XMLDOM.NEWDOMDOCUMENT('<ROOT>
    <A ATTR1="A_VALUE">
        <ACHILD>ACHILD TXT</ACHILD>
    </A>
    <B>B TXT</B>
    <C/>
  </ROOT>');
  FPATH := 'dir/simplexml.xml';
  DBE_XMLDOM.WRITETOFILE(DOC, FPATH);
END;
/
-- Expected result:
ANONYMOUS BLOCK EXECUTE

-- 2. Write an XML document to a specified file using the specified character set.
DECLARE
  SRC   VARCHAR(1000);
  FPATH VARCHAR2(1000);
```



```
  DOC   DBE_XMLDOM.DOMDOCUMENT;
  ELE   DBE_XMLDOM.DOMELEMENT;
BEGIN
  FPATH := 'dir/simplexml.xml';
  SRC := '<ROOT>
    <A ATTR1="A_VALUE">
        <ACHILD>ACHILD TXT</ACHILD>
    </A>
    <B>B TXT</B>
    <C/>
  </ROOT>';
  DOC := DBE_XMLDOM.NEWDOMDOCUMENT(SRC);
  ELE := DBE_XMLDOM.GETDOCUMENTELEMENT(DOC);
  DBE_XMLDOM.WRITETOFILE(DBE_XMLDOM.MAKENODE(ELE), FPATH, 'ASCII');
  DBE_XMLDOM.FREEDOCUMENT(DOC);
END;
/
-- Expected result:
ANONYMOUS BLOCK EXECUTE

-- Clean the environment.
drop directory dir;
```

- DBE_XMLDOM.GETSESSIONTREENUM

  Queries the number of DOM trees of all types in the current session. The prototype of the DBE_XMLDOM.GETSESSIONTREENUM function is as follows:

  ```
  DBE_XMLDOM.GETSESSIONTREENUM()
  RETURN INTEGER;
  ```

  NOTE

  For DOM trees that have used FREEElement and FREENode, this function still counts them.

  Example:

  ```
  -- Create three documents and obtain the number of DOM trees in the current session.
  DECLARE
    doc DBE_XMLDOM.DOMDocument;
    doc2 DBE_XMLDOM.DOMDocument;
    doc3 DBE_XMLDOM.DOMDocument;

    buffer varchar2(1010);
  BEGIN
  -- Create three documents.
    doc := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
  <root>
     <elem1 attr="attrtest">
        <elem2>Im text</elem2>
        <elem3>Im text too</elem3>
     </elem1>
     <elem4>Text</elem4>
  </root>
  ');
    doc2 := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
  <computer size="ITX" price="19999">
     <cpu>Ryzen 9 3950X</cpu>
     <ram>32GBx2 DDR4 3200MHz</ram>
     <motherboard>ROG X570i</motherboard>
     <gpu>RTX2070 Super</gpu>
     <ssd>1TB NVMe Toshiba + 2TB NVMe WD Black</ssd>
     <hdd>12TB WD Digital</hdd>
     <psu>CORSAIR SF750</psu>
     <case>LIANLI TU150</case>
  </computer>
  ');
    doc3 := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
  <bookstore>
     <book genre="autobiography" publicationdate="1981" ISBN="1-861003-11-0">
        <title>The Autobiography of Benjamin Franklin</title>
  ```



```
      <author>
        <first-name>Benjamin</first-name>
        <last-name>Franklin</last-name>
      </author>
      <price>8.99</price>
   </book>
   <book genre="novel" publicationdate="1967" ISBN="0-201-63361-2">
      <title>The Confidence Man</title>
      <author>
        <first-name>Herman</first-name>
        <last-name>Melville</last-name>
      </author>
      <price>11.99</price>
   </book>
   <book genre="philosophy" publicationdate="1991" ISBN="1-861001-57-6">
      <title>The Gorgias</title>
      <author>
        <name>Plato</name>
      </author>
      <price>9.99</price>
   </book>
</bookstore>
');
-- Print IDs.
  DBE_OUTPUT.PRINT_LINE(doc.id);
  DBE_OUTPUT.PRINT_LINE(doc2.id);
  DBE_OUTPUT.PRINT_LINE(doc3.id);
-- Call functions and print them.
  DBE_OUTPUT.PRINT_LINE(DBE_XMLDOM.GETSESSIONTREENUM());
-- Release the documents.
  DBE_XMLDOM.FREEDOCUMENT(doc);
  DBE_XMLDOM.FREEDOCUMENT(doc2);
  DBE_XMLDOM.FREEDOCUMENT(doc3);
END;
/
-- Expected result (if the XMLDOM API has been executed before the current session, the result is
uncertain):
00000000000000000200000001
01000000000000000300000001
02000000000000000400000001
3
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.GETDOCTREESINFO

  Queries the DOM tree information of the document type in the current session, such as the memory usage. The prototype of the DBE_XMLDOM.GETDOCTREESINFO function is as follows:

  ```
  DBE_XMLDOM.GETDOCTREESINFO()
  RETURN VARCHAR2;
  ```

  NOTE

  This function collects statistics only on DOM tree nodes of the document type.

  Example:

  ```
  -- Create three documents and obtain the information about the document tree in the current session.
  DECLARE
    doc DBE_XMLDOM.DOMDocument;
    doc2 DBE_XMLDOM.DOMDocument;
    doc3 DBE_XMLDOM.DOMDocument;

    buffer varchar2(1010);
  BEGIN
  -- Create three documents.
    doc := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
  <root>
     <elem1 attr="attrtest">
        <elem2>Im text</elem2>
        <elem3>Im text too</elem3>
  ```



```
    </elem1>
    <elem4>Text</elem4>
</root>
');
  doc2 := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
<computer size="ITX" price="19999">
    <cpu>Ryzen 9 3950X</cpu>
    <ram>32GBx2 DDR4 3200MHz</ram>
    <motherboard>ROG X570i</motherboard>
    <gpu>RTX2070 Super</gpu>
    <ssd>1TB NVMe Toshiba + 2TB NVMe WD Black</ssd>
    <hdd>12TB WD Digital</hdd>
    <psu>CORSAIR SF750</psu>
    <case>LIANLI TU150</case>
</computer>
');
  doc3 := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
<bookstore>
    <book genre="autobiography" publicationdate="1981" ISBN="1-861003-11-0">
        <title>The Autobiography of Benjamin Franklin</title>
        <author>
            <first-name>Benjamin</first-name>
            <last-name>Franklin</last-name>
        </author>
        <price>8.99</price>
    </book>
    <book genre="novel" publicationdate="1967" ISBN="0-201-63361-2">
        <title>The Confidence Man</title>
        <author>
            <first-name>Herman</first-name>
            <last-name>Melville</last-name>
        </author>
        <price>11.99</price>
    </book>
    <book genre="philosophy" publicationdate="1991" ISBN="1-861001-57-6">
        <title>The Gorgias</title>
        <author>
            <name>Plato</name>
        </author>
        <price>9.99</price>
    </book>
</bookstore>
');
-- Print IDs.
  DBE_OUTPUT.PRINT_LINE(doc.id);
  DBE_OUTPUT.PRINT_LINE(doc2.id);
  DBE_OUTPUT.PRINT_LINE(doc3.id);
-- Call functions and print them.
  DBE_OUTPUT.PRINT_LINE(DBE_XMLDOM.GETDOCTREESINFO());
-- Release the documents.
  DBE_XMLDOM.FREEDOCUMENT(doc);
  DBE_XMLDOM.FREEDOCUMENT(doc2);
  DBE_XMLDOM.FREEDOCUMENT(doc3);
END;
/
-- Expected result (if the XMLDOM API has been executed before the current session, the result is
uncertain):
00000000000000000200000001
01000000000000000300000001
02000000000000000400000001
|ID:00000000000000000200000001   |Node count:11   |Memory used:151 byte   |
|ID:01000000000000000300000001   |Node count:22   |Memory used:322 byte   |
|ID:02000000000000000400000001   |Node count:48   |Memory used:654 byte   |

ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.GETDETAILDOCTREEINFO

Queries the number of subnodes of each type in the transferred document. The prototype of the DBE_XMLDOM.GETDETAILDOCTREEINFO function is as follows:

```
DBE_XMLDOM.GETDETAILDOCTREEINFO(
   doc     IN      DOMDOCUMENT
)
RETURN VARCHAR2;
```

Table 10-462 DBE_XMLDOM.GETDETAILDOCTREEINFO parameters

Parameter	Description
doc	Specified DOMDocument node

NOTE

This function collects statistics only on DOM tree nodes of the document type.

Example:

```
-- Create three documents and use this function to obtain the number of nodes of each type in each
document.
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  doc2 DBE_XMLDOM.DOMDocument;
  doc3 DBE_XMLDOM.DOMDocument;

  buffer varchar2(1010);
BEGIN
-- Create three documents.
  doc := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
<root>
   <elem1 attr="attrtest">
      <elem2>Im text</elem2>
      <elem3>Im text too</elem3>
   </elem1>
   <elem4>Text</elem4>
</root>
');
  doc2 := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
<computer size="ITX" price="19999">
   <cpu>Ryzen 9 3950X</cpu>
   <ram>32GBx2 DDR4 3200MHz</ram>
   <motherboard>ROG X570i</motherboard>
   <gpu>RTX2070 Super</gpu>
   <ssd>1TB NVMe Toshiba + 2TB NVMe WD Black</ssd>
   <hdd>12TB WD Digital</hdd>
   <psu>CORSAIR SF750</psu>
   <case>LIANLI TU150</case>
</computer>
');
  doc3 := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0"?>
<bookstore>
   <book genre="autobiography" publicationdate="1981" ISBN="1-861003-11-0">
      <title>The Autobiography of Benjamin Franklin</title>
      <author>
         <first-name>Benjamin</first-name>
         <last-name>Franklin</last-name>
      </author>
      <price>8.99</price>
   </book>
   <book genre="novel" publicationdate="1967" ISBN="0-201-63361-2">
      <title>The Confidence Man</title>
      <author>
         <first-name>Herman</first-name>
```

```
            <last-name>Melville</last-name>
        </author>
        <price>11.99</price>
    </book>
    <book genre="philosophy" publicationdate="1991" ISBN="1-861001-57-6">
        <title>The Gorgias</title>
        <author>
            <name>Plato</name>
        </author>
        <price>9.99</price>
    </book>
</bookstore>
');
-- Print IDs.
  DBE_OUTPUT.PRINT_LINE(doc.id);
  DBE_OUTPUT.PRINT_LINE(doc2.id);
  DBE_OUTPUT.PRINT_LINE(doc3.id);
-- Call functions and print them.
  buffer := DBE_XMLDOM.GETDETAILDOCTREEINFO(doc);
  DBE_OUTPUT.PRINT_LINE(buffer);
  buffer := DBE_XMLDOM.GETDETAILDOCTREEINFO(doc2);
  DBE_OUTPUT.PRINT_LINE(buffer);
  buffer := DBE_XMLDOM.GETDETAILDOCTREEINFO(doc3);
  DBE_OUTPUT.PRINT_LINE(buffer);
-- Release the documents.
  DBE_XMLDOM.FREEDOCUMENT(doc);
  DBE_XMLDOM.FREEDOCUMENT(doc2);
  DBE_XMLDOM.FREEDOCUMENT(doc3);
END;
/
-- Expected result (if the XMLDOM API has been executed before the current session, the result is
uncertain):
00000000000000000200000001
01000000000000000300000001
02000000000000000400000001
|ID:00000000000000000200000001    |Element count:5    |Attribute count:1    |Text count:4    |
|ID:01000000000000000300000001    |Element count:9    |Attribute count:2    |Text count:10    |
|ID:02000000000000000400000001    |Element count:18    |Attribute count:9    |Text count:20    |
ANONYMOUS BLOCK EXECUTE
```

- DBE_XMLDOM.GETELEMENTSBYTAGNAME

  Searches for the DOMNodelist node by name in the XML file and returns the node. The prototype of the DBE_XMLDOM.GETELEMENTSBYTAGNAME function is:

  ```
  DBE_XMLDOM.GETELEMENTSBYTAGNAME(
    doc        IN      DOMDOCUMENT,
    tagname    IN      VARCHAR2)
  RETURN DOMNODELIST;
  ```

  Searches for the DOMNodelist node by name in the XML file and returns the node. The prototype of the DBE_XMLDOM.GETELEMENTSBYTAGNAME function is:

  ```
  DBE_XMLDOM.GETELEMENTSBYTAGNAME(
    elem       IN      DOMELEMENT,
    tagname       IN      VARCHAR2)
  RETURN DOMNODELIST;
  ```

  Searches for the DOMNodelist node by name and namespace in the XML file and returns the node. The prototype of the DBE_XMLDOM.GETELEMENTSBYTAGNAME function is:

  ```
  DBE_XMLDOM.GETELEMENTSBYTAGNAME(
    elem       IN      DOMELEMENT,
    tagname       IN      VARCHAR2,
    ns         IN      VARCHAR2)
   RETURN DOMNODELIST;
  ```


Table 10-463 DBE_XMLDOM.GETELEMENTSBYTAGNAME parameters

Parameter	Description
doc	Specified DOMDocument node
elem	Specified DOMElement node
tagname	Tag name. Using the wildcard (*) will match any tag.
ns	Namespace. Using the wildcard (*) will match any namespace.

Example:

```
-- 1. Search the DOMDocument node by TAGNAME and return the matched DOMNodelist node list.
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  root_elem DBE_XMLDOM.DOMElement;
  child_node DBE_XMLDOM.DOMNODE;
  node_list DBE_XMLDOM.DOMNODELIST;
  buffer VARCHAR2(1000);
BEGIN
  doc := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0" encoding="UTF-8"?>
<computer size="ITX" price="19999">
    <cpu>Ryzen 9 3950X</cpu>
    <cpu>Ryzen 9 5950X_1</cpu>
    <ram>32GBx2 DDR4 3200MHz<cpu>Ryzen <cpu>Ryzen 9 5950X_2</cpu></cpu></ram>
    <motherboard>ROG X570i</motherboard>
    <gpu>RTX2070 Super</gpu>
    <ssd>1TB NVMe Toshiba + 2TB NVMe WD Black</ssd>
    <hdd>12TB WD Digital</hdd>
    <psu>CORSAIR SF750</psu>
    <case>LIANLI TU150</case>
</computer>');
  root_elem := DBE_XMLDOM.GETDOCUMENTELEMENT(doc);
  node_list := DBE_XMLDOM.GETELEMENTSBYTAGNAME(doc, 'cpu');
  child_node := DBE_XMLDOM.ITEM(node_list, 2);
  DBE_XMLDOM.WRITETOBUFFER(child_node, buffer);
  DBE_OUTPUT.PRINT_LINE(buffer);
END;
/
-- Expected result:
<cpu>Ryzen <cpu>Ryzen 9 5950X_2</cpu></cpu>
ANONYMOUS BLOCK EXECUTE

-- 2. Search the DOMElement node by TAGNAME and return the matched DOMNodelist node list.
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  root_elem DBE_XMLDOM.DOMElement;
  child_node DBE_XMLDOM.DOMNODE;
  node_list DBE_XMLDOM.DOMNODELIST;
  buffer VARCHAR2(1000);
BEGIN
  doc := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0" encoding="UTF-8"?>
<computer size="ITX" price="19999">
    <cpu>Ryzen 9 3950X</cpu>
    <cpu>Ryzen 9 5950X_1</cpu>
    <ram>32GBx2 DDR4 3200MHz<cpu>Ryzen 9 5950X_2<cpu>Ryzen 9 5950X_3<cpu>Ryzen 9
5950X_4</cpu></cpu></cpu></ram>
    <motherboard>ROG X570i</motherboard>
    <gpu>RTX2070 Super</gpu>
    <ssd>1TB NVMe Toshiba + 2TB NVMe WD Black</ssd>
    <hdd>12TB WD Digital</hdd>
```



```
    <psu>CORSAIR SF750</psu>
    <case>LIANLI TU150</case>
</computer>');
  root_elem := DBE_XMLDOM.GETDOCUMENTELEMENT(doc);
  node_list := DBE_XMLDOM.GETELEMENTSBYTAGNAME(root_elem, 'cpu');
  child_node := DBE_XMLDOM.ITEM(node_list, 3);
  DBE_XMLDOM.WRITETOBUFFER(child_node, buffer);
  DBE_OUTPUT.PRINT_LINE(buffer);
END;
/
-- Expected result:
<cpu>Ryzen 9 5950X_3<cpu>Ryzen 9 5950X_4</cpu></cpu>
ANONYMOUS BLOCK EXECUTE

-- 3. Search the DOMElement node by TAGNAME and NAMESPACE, and return the matched
DOMNodelist node list.
DECLARE
  doc DBE_XMLDOM.DOMDocument;
  root_elem DBE_XMLDOM.DOMElement;
  child_node DBE_XMLDOM.DOMNODE;
  node_list DBE_XMLDOM.DOMNODELIST;
  buffer VARCHAR2(1000);
BEGIN
  doc := DBE_XMLDOM.NEWDOMDOCUMENT('<?xml version="1.0" encoding="UTF-8"?>
<computer size="ITX" price="19999" xmlns:h="www.huawei.com">
    <cpu>Ryzen 9 3950X</cpu>
    <cpu>Ryzen 9 5950X_1</cpu>
    <h:cpu>ns Ryzen 9 5950X_2</h:cpu>
    <ram>32GBx2 DDR4 3200MHz<cpu>Ryzen 9 5950X_3<cpu>Ryzen 9 5950X_4<cpu>Ryzen 9
5950X_5</cpu></cpu></cpu></ram>
    <motherboard>ROG X570i</motherboard>
    <gpu>RTX2070 Super</gpu>
    <ssd>1TB NVMe Toshiba + 2TB NVMe WD Black</ssd>
    <hdd>12TB WD Digital</hdd>
    <psu>CORSAIR SF750</psu>
    <case>LIANLI TU150</case>
</computer>');
  root_elem := DBE_XMLDOM.GETDOCUMENTELEMENT(doc);
  node_list := DBE_XMLDOM.GETELEMENTSBYTAGNAME(root_elem, 'cpu', 'www.huawei.com');
  child_node := DBE_XMLDOM.ITEM(node_list, 0);
  DBE_XMLDOM.WRITETOBUFFER(child_node, buffer);
  DBE_OUTPUT.PRINT_LINE(buffer);
END;
/
-- Expected result:
<h:cpu>ns Ryzen 9 5950X_2</h:cpu>
ANONYMOUS BLOCK EXECUTE
```

10.12.2.20 DBE_XMLPARSER

API Description

The DBE_XMLPARSER API is used to deserialize XML character strings and convert the character strings that store XML documents to document nodes. **Table 10-464** lists all APIs supported by the DBE_XMLPARSER advanced package.

The XMLParser data type can be used to store XMLParser data. The maximum number of XMLParser data records that can be stored is 16777215. The XMLPARSER data type can parse and create the DOMDocument node according to the input character string. The advanced package also provides the corresponding set and get APIs to perform operations on the constraint attributes of the parsing process.

NOTE

When the DBE_XMLPARSER advanced package is used in the database whose character set is set to **SQL_ASCII**, an error is reported if characters beyond the ASCII range are input.

The DBE_XMLPARSER advanced package supports only the A-compatible mode.

Table 10-464 DBE_XMLPARSER parameters

API	Description
DBE_XMLPARSER.FREEPARSER	Frees a parser.
DBE_XMLPARSER.GETDOCUMENT	Obtains the parsed document node.
DBE_XMLPARSER.GETVALIDATIONMODE	Obtains the validation attribute.
DBE_XMLPARSER.NEWPARSER	Creates a parser instance.
DBE_XMLPARSER.PARSEBUFFER	Parses the VARCHAR string.
DBE_XMLPARSER.PARSECLOB	Parses the CLOB string.
DBE_XMLPARSER.SETVALIDATIONMODE	Sets the validation attribute.

- DBE_XMLPARSER.FREEPARSER

  Frees a given parser object.

  The stored procedure prototype of DBE_XMLPARSER.FREEPARSER is as follows:

```
DBE_XMLPARSER.FREEPARSER (
  p     IN     parser);
```

  Table 10-465 DBE_XMLPARSER.FREEPARSER parameters

Parameter	Description
p	Parser object

  Example:

```
-- Create a parser and then release it.
DECLARE
  l_parser dbe_xmlparser.parser;
  BEGIN
  l_parser := dbe_xmlparser.newparser;
-- Directly release the l_parser instance.
  dbe_xmlparser.freeparser(l_parser);
END;
/
```

  Result: The operation is successful.

- DBE_XMLPARSER.GETDOCUMENT

  Returns the root node of the DOM tree document constructed by the parser. This function can be called only after the document is parsed.

The prototype of the DBE_XMLPARSER.GETDOCUMENT function is as follows:

```
DBE_XMLPARSER.GETDOCUMENT (
 p     IN     parser)
RETURN DOMDocument;
```

Table 10-466 DBE_XMLPARSER.GETDOCUMENT parameters

Parameter	Description
p	Parser object

NOTE

- If the GETDOCUMENT function has no input parameter, an error is reported.
- If the **parser** parameter of the GETDOCUMENT function is null, **NULL** is returned.
- If the parser input by the GETDOCUMENT function has not parsed any document, **NULL** is returned.

Example:

```
-- Create a parser to parse character strings and print the obtained document.
DECLARE
  l_parser dbe_xmlparser.parser;
  l_doc dbe_xmldom.domdocument;
  buffer varchar2 :=
'<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Donot forget me this weekend!</body>
</note>';
  buffer2 varchar2;
  BEGIN
  l_parser := dbe_xmlparser.newparser;
-- The l_parser parses the character string and obtains the DOMDocument node through
GETDOCUMENT.
  dbe_xmlparser.PARSEBUFFER(l_parser, buffer);
  l_doc := dbe_xmlparser.getdocument(l_parser);
-- Print the content in l_doc.
  dbe_xmldom.writetobuffer(l_doc, buffer2);
  RAISE NOTICE '%', buffer2;

  dbe_xmlparser.freeparser(l_parser);
  dbe_xmldom.freedocument(l_doc);
END;
/
```

Execution result:

```
NOTICE:  <?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Donot forget me this weekend!</body>
</note>
```

- DBE_XMLPARSER.GETVALIDATIONMODE

  Obtains the parsing validation mode of a specified parser. If DTD validation is enabled, **TRUE** is returned. Otherwise, **FALSE** is returned.

  The prototype of the DBE_XMLPARSER.GETVALIDATIONMODE function is as follows:


```
DBE_XMLPARSER.GETVALIDATIONMODE (
 p     IN     parser)
 RETURN BOOLEAN;
```

Table 10-467 DBE_XMLPARSER.GETVALIDATIONMODE parameters

Parameter	Description
p	Parser object

Example:

```
-- Create a parser and use GETVALIDATIONMODE to check whether the parser validation mode is
enabled.
DECLARE
  l_parser dbe_xmlparser.parser;
BEGIN
  l_parser := dbe_xmlparser.newparser();
    if (dbe_xmlparser.GETVALIDATIONMODE(l_parser) = true) then
 RAISE NOTICE 'validation';
    else
    RAISE NOTICE 'no validation';
  end if;
  dbe_xmlparser.freeparser(l_parser);
END;
/
```

Execution result:

```
NOTICE:  validation
```

- DBE_XMLPARSER.NEWPARSER

  Creates a parser object and returns a new parser instance.

  The prototype of the DBE_XMLPARSER.NEWPARSER function is as follows:

  ```
  DBE_XMLPARSER.NEWPARSER
   RETURN Parser;
  ```

  Example:

  ```
  -- Create a parser to parse character strings and then free the parser.
  DECLARE
    -- Create a parser.
    l_parser dbe_xmlparser.parser;
    l_doc dbe_xmldom.domdocument;
    buffer varchar2(1000) :=
      '<?xml version="1.0" encoding="UTF-8"?>
      <note>
      <to>Tove</to>
      <from>Jani</from>
      <heading>Reminder</heading>
      <body>Donot forget me this weekend!</body>
      </note>';
    buffer2 varchar2(1000);
    BEGIN
    l_parser := dbe_xmlparser.newparser;
    -- Parse the document and create a new DOM document.
    dbe_xmlparser.PARSEBUFFER(l_parser, buffer);

    dbe_xmlparser.freeparser(l_parser);
  END;
  /
  ```

  Result: The operation is successful.

- DBE_XMLPARSER.PARSEBUFFER

  Parses XML documents stored in strings.

The stored procedure prototype of DBE_XMLPARSER.PARSEBUFFER is as follows:

```
DBE_XMLPARSER.PARSEBUFFER (
  p     IN     parser,
  doc   IN VARCHAR2);
```

Table 10-468 DBE_XMLPARSER.PARSEBUFFER parameters

Parameter	Description
p	Parser object
doc	A string that stores XML documents

NOTE

- The maximum length of a character string that can be parsed by the PARSEBUFFER function is 32767. If the length exceeds the maximum, an error is reported.
- Different from the database A, this database supports only UTF-8 in terms of character encoding, and **version** can only be set to **1.0**. If versions 1.0 to 1.9 are parsing, a warning appears but the execution is normal. For versions later than 1.9, an error is reported.
- DTD validation differences:
  - **!ATTLIST to type (CHECK|check|Check) "Ch..."** reports an error because the default value **"Ch..."** is not an enumerated value in the brackets. However, the database A does not report this error.
  - **<!ENTITY baidu "www.baidu.com">..... &Baidu;&writer** reports an error because the letters are case sensitive. **Baidu** cannot correspond to **baidu**. However, the database A does not report this error.
- Namespace validation difference: Undeclared namespace tags are parsed. However, the database A reports an error.
- Difference in parsing XML predefined entities: **'** and **"** are parsed and translated into ' and ". However, predefined entities in database A are not translated into characters.

Example:

```
-- Create a parser to parse character strings and print the obtained document.
DECLARE
  l_parser dbe_xmlparser.parser;
  l_doc dbe_xmldom.domdocument;
  buffer varchar2 :=
'<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Donot forget me this weekend!</body>
</note>';
  buffer2 varchar2;
  BEGIN
  l_parser := dbe_xmlparser.newparser;
  dbe_xmlparser.PARSEBUFFER(l_parser, buffer);
  l_doc := dbe_xmlparser.getdocument(l_parser);

  dbe_xmldom.writetobuffer(l_doc, buffer2);
  RAISE NOTICE '%', buffer2;

  dbe_xmlparser.freeparser(l_parser);
  dbe_xmldom.freedocument(l_doc);
```



```
END;
/
```

Execution result:

```
NOTICE:  <?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Donot forget me this weekend!</body>
</note>
```

- DBE_XMLPARSER.PARSECLOB

  Parses XML documents stored in a CLOB.

  The stored procedure prototype of DBE_XMLPARSER.PARSECLOB is as follows:

```
DBE_XMLPARSER.PARSECLOB (
  p     IN     parser,
  doc   IN CLOB);
```

Table 10-469 DBE_XMLPARSER.PARSECLOB parameters

Parameter	Description
p	Parser object
doc	A CLOB that stores XML documents

NOTE

- PARSECLOB cannot parse CLOBs greater than or equal to 2 GB.
- Different from the database A, this database supports only UTF-8 in terms of character encoding, and **version** can only be set to **1.0**. If versions 1.0 to 1.9 are parsing, a warning appears but the execution is normal. For versions later than 1.9, an error is reported.
- DTD validation differences:
  - **!ATTLIST to type (CHECK|check|Check) "Ch..."** reports an error because the default value **"Ch..."** is not an enumerated value in the brackets. However, the database A does not report this error.
  - **<!ENTITY baidu "www.baidu.com">..... &Baidu;&writer** reports an error because the letters are case sensitive. **Baidu** cannot correspond to **baidu**. However, the database A does not report this error.
- Namespace validation difference: Undeclared namespace tags are parsed. However, the database A reports an error.
- Difference in parsing XML predefined entities: **'** and **"** are parsed and translated into ' and ". However, predefined entities in database A are not translated into characters.

Example:

```
-- Create a parser to parse character strings and print the obtained document.
DECLARE

 l_clob clob :=
'<?xml version="1.0" encoding="UTF-8"?>
 <note>
 <to>Tove</to>
 <from>Jani</from>
 <heading>Reminder</heading>
 <body>this weekend!</body>
 </note>';
```



```
-- Create a parser.
l_parser dbe_xmlparser.parser;
l_doc dbe_xmldom.domdocument;
buffer varchar2(1000);
BEGIN
l_parser := dbe_xmlparser.newparser;
-- Parse the document and create a new DOM document.
dbe_xmlparser.parseclob(l_parser, l_clob);
l_doc := dbe_xmlparser.getdocument(l_parser);
dbe_xmldom.writetobuffer(l_doc, buffer);
RAISE NOTICE '%',buffer;

dbe_xmlparser.freeparser(l_parser);
dbe_xmldom.freedocument(l_doc);

END;
/
```

Execution result:

```
NOTICE:  <?xml version="1.0" encoding="UTF-8"?>
<note>
 <to>Tove</to>
 <from>Jani</from>
 <heading>Reminder</heading>
 <body>this weekend!</body>
</note>
```

- DBE_XMLPARSER.SETVALIDATIONMODE

  Sets the parsing validation mode of a specified parser.

  The stored procedure prototype of DBE_XMLPARSER.SETVALIDATIONMODE is as follows:

```
DBE_XMLPARSER.SETVALIDATIONMODE(
 p     IN     parser)
 yes   IN BOOLEAN);
```

Table 10-470 DBE_XMLPARSER.SETVALIDATIONMODE parameters

Parameter	Description
p	Parser object
yes	Mode to be set: • **TRUE**: DTD validation is enabled. • **FALSE**: DTD validation is disabled.

NOTE

- If the input parameter **yes** of the SETVALIDATIONMODE function is null, the parsing validation mode of the parser is not changed.
- By default, the DTD validation is enabled during parser initialization.

Example 1:

```
-- Create a parser. The XML character string to be parsed does not match the DTD format.
-- If setValidationMode is set to false, the string can be parsed. If setValidationMode is set to true, an error is reported during parsing.
DECLARE
 l_clob clob :=
'<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
```



```
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<table>
<name attr1="WEB" attr2="web2">African Coffee Table</name>
<width>80</width>
<length>120</length>
</table>';
 l_parser dbe_xmlparser.parser;
 l_doc dbe_xmldom.domdocument;
buffer varchar2(1000);
 BEGIN
 l_parser := dbe_xmlparser.newparser;
 -- Set it to false for parsing.
   dbe_xmlparser.setValidationMode(l_parser, false);
   dbe_xmlparser.parseclob(l_parser, l_clob);
   l_doc := dbe_xmlparser.getdocument(l_parser);
   dbe_xmldom.writetobuffer(l_doc, buffer);
   RAISE NOTICE '%', buffer;
   dbe_xmlparser.freeparser(l_parser);
   dbe_xmldom.freedocument(l_doc);
 END;
 /
```

Execution result:

```
NOTICE:  <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note [
<!ELEMENT note (to , from , heading , body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<table>
 <name attr1="WEB" attr2="web2">African Coffee Table</name>
 <width>80</width>
 <length>120</length>
 </table>
```

Example 2:

```
-- Create a parser. The XML character string to be parsed does not match the DTD format.
-- An error is reported during parsing after setValidationMode is set to true.
DECLARE
  l_clob clob :=
 '<!DOCTYPE note [
 <!ELEMENT note (to,from,heading,body)>
 <!ELEMENT to (#PCDATA)>
 <!ELEMENT from (#PCDATA)>
 <!ELEMENT heading (#PCDATA)>
 <!ELEMENT body (#PCDATA)>
 ]>
 <table>
 <name attr1="WEB" attr2="web2">African Coffee Table</name>
 <width>80</width>
 <length>120</length>
 </table>';
 l_parser dbe_xmlparser.parser;
 l_doc dbe_xmldom.domdocument;
buffer varchar2(1000);
 BEGIN
 l_parser := dbe_xmlparser.newparser;
 -- Set it to true for parsing.
--The XML character string does not match the DTD format. An error is expected.
   dbe_xmlparser.setValidationMode(l_parser, true);
   dbe_xmlparser.parseclob(l_parser, l_clob);
   l_doc := dbe_xmlparser.getdocument(l_parser);
   dbe_xmldom.writetobuffer(l_doc, buffer);
   dbe_xmlparser.freeparser(l_parser);
   dbe_xmldom.freedocument(l_doc);
```



```
END;
/
```

Execution result:

```
An error is reported during xmlparser parsing.
ERROR:  invalid XML document
```

10.12.2.21 DBE_DESCRIBE

Data Types

The advanced package **DBE_DESCRIBE** has two built-in data types. The two data types are created using user-defined types and are used for the return values of the DESCRIBE_PROCEDURE API.

- DBE_DESCRIBE.NUMBER_TABLE

  This type is the TABLE type of NUMBER and is implemented through the TABLE OF syntax.

  The prototype of the DBE_DESCRIBE.NUMBER_TABLE type is as follows:

  ```
  CREATE TYPE DBE_DESCRIBE.NUMBER_TABLE AS TABLE OF NUMBER INDEX BY INTEGER;
  ```

- DBE_DESCRIBE.VARCHAR2_TABLE

  This type is the TABLE type of VARCHAR2 and is implemented through the TABLE OF syntax.

  The prototype of the DBE_DESCRIBE.VARCHAR2_TABLE type is as follows:

  ```
  CREATE TYPE DBE_DESCRIBE.VARCHAR2_TABLE AS TABLE OF VARCHAR2(30) INDEX BY INTEGER;
  ```

API Description

The advanced package **DBE_DESCRIBE** provides the DESCRIBE_PROCEDURE API to return the parameter information (such as parameter names and parameter data types) of functions or stored procedures in a list.

Table 10-471 DBE_DESCRIBE overview

API	Description
DBE_DESCRIBE.DESCRIBE_PROCEDURE	Displays the parameters of a stored procedure or function.

- DBE_DESCRIBE.DESCRIBE_PROCEDURE

  The stored procedure DESCRIBE_PROCEDURE is used to display the parameter information of a stored procedure or function, such as the parameter name, parameter mode, and parameter location. The parameter information of a function or stored procedure is returned in a list.

  The prototype of the DBE_DESCRIBE.DESCRIBE_PROCEDURE function is as follows:

  ```
  DBE_DESCRIBE.DESCRIBE_PROCEDURE(
    object_name               IN  VARCHAR2,
    reserved1                 IN  VARCHAR2,
    reserved2                 IN  VARCHAR2,
    overload                  OUT DBE_DESCRIBE.NUMBER_TABLE,
  ```



```
 dataposition              OUT DBE_DESCRIBE.NUMBER_TABLE,
 datalevel                 OUT DBE_DESCRIBE.NUMBER_TABLE,
 argument_name             OUT DBE_DESCRIBE.VARCHAR2_TABLE,
 datatype                  OUT DBE_DESCRIBE.NUMBER_TABLE,
 default_value             OUT DBE_DESCRIBE.NUMBER_TABLE,
 in_out                    OUT DBE_DESCRIBE.NUMBER_TABLE,
 datalength                OUT DBE_DESCRIBE.NUMBER_TABLE,
 dataprecision             OUT DBE_DESCRIBE.NUMBER_TABLE,
 scale                     OUT DBE_DESCRIBE.NUMBER_TABLE,
 radix                     OUT DBE_DESCRIBE.NUMBER_TABLE,
 spare                     OUT DBE_DESCRIBE.NUMBER_TABLE,
 include_string_constraints OUT BOOLEAN
);
```

Table 10-472 DBE_DESCRIBE.DESCRIBE_PROCEDURE API

Parameter	Type	Whether NULL Is Allowed	Description
object_name	varchar2	No	Stored procedure name. The syntax format of this parameter is [[*schema*.]*package*.]*function*[@*dblink*], in which: • *schema* (optional): schema name. • *package* (optional): package name. • *function* (mandatory): name of a function or stored procedure. • *dblink* (optional): remote connection name.
reserved1	varchar2	Yes	Reserved parameter.
reserved2	varchar2	Yes	Reserved parameter.
overload	number_table	Yes	Unique ID assigned to a stored procedure or function. If a stored procedure or function is overloaded, overload is each overload of the stored procedure or function. Overload is numbered from 1 in ascending order based on the entire stored procedure or function. If a stored procedure or function is not overloaded, **overload** is set to **0**.
dataposition	number_table	Yes	Position of a specified parameter in the parameter list.
datalevel	number_table	Yes	It is set to **0**.

Parameter	Type	Whether NULL Is Allowed	Description
argument_name	varchar2_table	Yes	Name of the parameter associated with the specified stored procedure.
datatype	number_table	Yes	OID of the data type of the specified parameter.
default_value	number_table	Yes	If the specified parameter has a default value, the value is **1**. Otherwise, the value is **0**.
in_out	number_table	Yes	Parameter mode. The options are as follows: • **0**: IN • **1**: OUT • **2**: IN OUT
datalength	number_table	Yes	Not supported. Set it to **0** by default.
dataprecision	number_table	Yes	Not supported. Set it to **0** by default.
scale	number_table	Yes	Not supported. Set it to **0** by default.
radix	number_table	Yes	If the value is of the numeric type (such as **NUMBER** and **INTEGER**), **10** is returned. Otherwise, **0** is returned. For details about the numeric type, see **numeric types**.
spare	number_table	Yes	Reserved parameter. Set it to **0** by default.
include_string_constraints	Boolean	Yes	Reserved parameter, which does not need to be processed.

NOTE

- The data type of the **datatype** parameter is different from that of the A-compatible database. The GaussDB returns the OID of the data type, and the A-compatible database returns the number of the data type in the A-compatible database.
- The **include_string_constraints** parameter does not take effect on the stored procedure. The value of the **include_string_constraints** parameter does not change, and the return values of other parameters are not affected.
- The OIDs of the data types created by the create type operation are uncertain. Therefore, do not use these OIDs for fixed judgment.
- For the **dataposition** parameter, if a stored procedure is specified, the return value starts from **1**. If a function is specified, the return value starts from **0**. **0** indicates the position sequence number of the return value of the function.
- For the **argument_name** parameter, if a function is specified, the first position of the return value is empty. This position indicates the name of the return value of the described function (that is, the empty name).
- Do not directly specify a package. Otherwise, an error is reported.
- If you do not have the execute permission on a stored procedure, function, or package, the system considers that the stored procedure, function, or package does not exist and reports an error.
- The input parameters **reserved1** and **reserved2** are not involved in internal processing. Entering any character string does not affect the returned result.
- The advanced package cannot specify the stored procedures or functions obtained through DBLINK.
- You are advised to add the schema prefix before the specified stored procedure or function. If the schema prefix is omitted, the advanced package uses the schema of the current session to search for the entity to which the advanced package belongs. In this case, you need to change the value of **behavior_compat_options** to **bind_procedure_searchpath** for the advanced package to take effect.
- If the %type operation is used to obtain the data type from a table column, the constraint on the data type (such as NUMBER(3) and VARCHAR2(10)) is not retained.

Example:

```
-- Create a stored procedure to encapsulate the advanced package for printing return values.
CREATE PROCEDURE PRINT_DESCRIBE (obj_name IN VARCHAR2)
AS
  a_overload      DBE_DESCRIBE.NUMBER_TABLE;
  a_position      DBE_DESCRIBE.NUMBER_TABLE;
  a_level         DBE_DESCRIBE.NUMBER_TABLE;
  a_arg_name      DBE_DESCRIBE.VARCHAR2_TABLE;
  a_dty           DBE_DESCRIBE.NUMBER_TABLE;
  a_def_val       DBE_DESCRIBE.NUMBER_TABLE;
  a_mode          DBE_DESCRIBE.NUMBER_TABLE;
  a_length        DBE_DESCRIBE.NUMBER_TABLE;
  a_precision     DBE_DESCRIBE.NUMBER_TABLE;
  a_scale         DBE_DESCRIBE.NUMBER_TABLE;
  a_radix         DBE_DESCRIBE.NUMBER_TABLE;
  a_spare         DBE_DESCRIBE.NUMBER_TABLE;
  a_include_string_constraints BOOLEAN;
BEGIN
  DBE_DESCRIBE.DESCRIBE_PROCEDURE(
    obj_name,
    null,
    null,
    a_overload,
    a_position,
    a_level,
    a_arg_name,
    a_dty,
```

```
  a_def_val,
  a_mode,
  a_length,
  a_precision,
  a_scale,
  a_radix,
  a_spare,
  a_include_string_constraints
);
dbe_output.print('overload                ' || chr(9));
for indx in 1 .. a_overload.count
loop
  dbe_output.print(a_overload(indx) || chr(9));
end loop;

dbe_output.print_line(' ');

dbe_output.print('dataposition            ' || chr(9));
for indx in 1 .. a_position.count
loop
  dbe_output.print(a_position(indx) || chr(9));
end loop;

dbe_output.print_line(' ');

dbe_output.print('datalevel              ' || chr(9));
for indx in 1 .. a_level.count
loop
  dbe_output.print(a_level(indx) || chr(9));
end loop;

dbe_output.print_line(' ');

dbe_output.print('argument_name          ' || chr(9));
for indx in 1 .. a_arg_name.count
loop
  dbe_output.print(a_arg_name(indx) || chr(9));
end loop;

dbe_output.print_line(' ');

dbe_output.print('default_value          ' || chr(9));
for indx in 1 .. a_def_val.count
loop
  dbe_output.print(a_def_val(indx) || chr(9));
end loop;

dbe_output.print_line(' ');

dbe_output.print('in_out                 ' || chr(9));
for indx in 1 .. a_mode.count
loop
  dbe_output.print(a_mode(indx) || chr(9));
end loop;

dbe_output.print_line(' ');

dbe_output.print('length                 ' || chr(9));
for indx in 1 .. a_length.count
loop
  dbe_output.print(a_length(indx) || chr(9));
end loop;

dbe_output.print_line(' ');

dbe_output.print('precision              ' || chr(9));
for indx in 1 .. a_precision.count
loop
  dbe_output.print(a_precision(indx) || chr(9));
```



```
  end loop;

  dbe_output.print_line(' ');

  dbe_output.print('scale                    ' || chr(9));
  for indx in 1 .. a_scale.count
  loop
    dbe_output.print(a_scale(indx) || chr(9));
  end loop;

  dbe_output.print_line(' ');

  dbe_output.print('radix                    ' || chr(9));
  for indx in 1 .. a_radix.count
  loop
    dbe_output.print(a_radix(indx) || chr(9));
  end loop;

  dbe_output.print_line(' ');
END;
/

-- Create a function with three overloads.
CREATE OR REPLACE FUNCTION TEST_FUNC_OVERLOAD (
  param_a IN NUMBER,
  param_b IN VARCHAR2,
  param_c OUT TEXT
)
RETURN VARCHAR2 package
AS
BEGIN
  dbe_output.print_line('This procedure/function test num param.');
  RETURN 'This procedure/function test num param.';
END;
/

CREATE OR REPLACE FUNCTION TEST_FUNC_OVERLOAD (
  param_a IN NUMBER DEFAULT 20,
  param_b VARCHAR2 DEFAULT 'n',
  param_c IN TEXT, param_d OUT DATE,
  param_e INOUT RAW
)
RETURN VARCHAR2 package
AS
BEGIN
  dbe_output.print_line('This procedure/function test num param.');
  RETURN 'This procedure/function test num param.';
END;
/

CREATE OR REPLACE FUNCTION TEST_FUNC_OVERLOAD (
  param_a IN NUMBER DEFAULT 20,
  param_b VARCHAR2 DEFAULT 'n',
  param_c IN TEXT, param_d IN DATE,
  param_e OUT RAW, param_f INOUT INTEGER
)
RETURN VARCHAR2 package
AS
BEGIN
  dbe_output.print_line('This procedure/function test num param.');
  RETURN 'This procedure/function test num param.';
END;
/

-- Call the preceding encapsulation.
BEGIN PRINT_DESCRIBE('TEST_FUNC_OVERLOAD'); END;
/

-- Clean up.
```



```
DROP FUNCTION TEST_FUNC_OVERLOAD (
  param_a IN NUMBER,
  param_b IN VARCHAR2,
  param_c OUT TEXT
);

DROP FUNCTION TEST_FUNC_OVERLOAD (param_a IN NUMBER,
  param_b VARCHAR2,
  param_c IN TEXT,
  param_d OUT DATE,
  param_e INOUT RAW
);

DROP FUNCTION TEST_FUNC_OVERLOAD (param_a IN NUMBER,
  param_b VARCHAR2,
  param_c IN TEXT,
  param_d IN DATE,
  param_e OUT RAW,
  param_f INOUT INTEGER
);

DROP PROCEDURE PRINT_DESCRIBE (obj_name IN VARCHAR2);
```

10.12.2.22 prvt_ilm

The prvt_ilm API is an internal API of the ILM feature and cannot be directly called by users. Only the API names are listed and the API prototypes are not described.

API	Description
be_active_ado_window	The maintenance window updates the ilmadowindow triggering time and action.
be_create_ado_window_for_each_db	The maintenance window creates ilmadowindow in each database of the instance.
be_execute_ilm	The ilmadowind performs automatic evaluation.
flush_task_executestate	The automatic scheduling task updates the task status.
evaluate_obj_policy	Data objects are evaluated after scheduling is triggered.
change_be_ilm	dbe_ilm_admin.enable_ilm() and disable_ilm() help functions. This operation affects the overall ILM background scheduling of the cluster.
get_compression_ratio	dbe_compression.get_compression_ratio() help function.
get_compression_type	dbe_compression.get_compression_type() help function.
get_lastmodified_time	dbe_heat_map.row_heat_map() help function.
gs_ilm_ticker	The maintenance window performs an action, performs a dotting operation, and records the mapping between the LSN and time.
compress_blocks	prvt_ilm.ilm_job_action() help function.

API	Description
get_job_status	dbe_ilm.stop_ilm help function.
ilm_job_action	Executor of a compression task.
query_unfinishedjob_num	Queries whether there are jobs that are not complete, including jobs in the initial and running states.
ilm_seq_nextval	Obtains the next sequence value of the ILM.
ilm_seq_setval	Sets the current sequence value of the ILM.

10.12.2.23 DBE_XMLGEN

API Description

The **DBE_XMLGEN** system package converts the result of an SQL query into a standard XML format and returns the result. For details about all supported APIs, see **Table 2 DBE_XMLGEN**.

Table 10-473 DBE_XMLGEN data type

Type	Description
DBE_XMLGEN.CTXHANDLE	Data type used to store the XML output status.

NOTE

1. A maximum of 65,535 context handles can exist in a session. Closing the context handle does not reclaim the number.
2. The case of the table column and type in the output XML file is the same as that of the table field and type created by the user. If you need to write a large field and type name, use double quotation marks ("") to enclose the field and type name.
3. During NEWCONTEXTFROMHIERARCHY initialization, the SETNULLHANDLING, USENULLATTRIBUTEINDICATOR, and SETCONVERTSPECIALCHARS methods are used to set this parameter, but the setting does not take effect.

Table 10-474 DBE_XMLGEN

API	Description
DBE_XMLGEN.CONVERT	Encodes or decodes the input character string in XML format.
DBE_XMLGEN.NEWCONTEXT	Initializes the common context handle.

API	Description
DBE_XMLGEN.NEWCONTEXTFROMHIERARCHY	Initializes the context handle with recursive elements.
DBE_XMLGEN.SETCONVERTSPECIALCHARS	Specifies whether the output XML file needs to be encoded.
DBE_XMLGEN.SETNULLHANDLING	Sets how to display the null value in the XML file.
DBE_XMLGEN.SETROWSETTAG	Sets the name of the XML root node.
DBE_XMLGEN.SETROWTAG	Sets the tag name of each row of data in the XML file.
DBE_XMLGEN.USENULLATTRIBUTEINDICATOR	Adds the **xsi:nil="true"** attribute to the element where the **null** value is located in the XML file.
DBE_XMLGEN.USEITEMTAGSFORCOLL	Adds the suffix **'_item'** to the element where the variable of the array type is located.
DBE_XMLGEN.GETNUMROWSPROCESSED	Views the number of data rows returned by getxml or getxmltype last time.
DBE_XMLGEN.SETMAXROWS	Sets the maximum number of rows returned by getxml.
DBE_XMLGEN.SETSKIPROWS	Sets the number of SQL rows to be skipped.
DBE_XMLGEN.RESTARTQUERY	Restarts the SQL.
DBE_XMLGEN.GETXMLTYPE	Returns the XML text of the XMLTYPE type.
DBE_XMLGEN.GETXML	Returns the XML text of the CLOB type.
DBE_XMLGEN.CLOSECONTEXT	Disables the context handle.

- DBE_XMLGEN.CONVERT

  Encodes or decodes the input character string in XML format.

  The conversion is performed according to the following rules.

  Table 10-475 XML encoding rules

Original Value	Code Value
&	&

Original Value	Code Value
<	<
>	>
"	"
'	'

Function prototype:

```
DBE_XMLGEN.CONVERT(XMLSTR IN VARCHAR2, FLAG IN NUMBER := 0) RETURNS VARCHAR2;
DBE_XMLGEN.CONVERT(XMLCLOB IN CLOB, FLAG IN NUMBER := 0) RETURNS CLOB;
```

Parameter description

Table 10-476 DBE_XMLGEN.CONVERT parameters

Parameter	Description
XMLSTR	XML character string to be converted. The value is of the VARCHAR2 type.
XMLCLOB	XML character string to be converted. The value is of the CLOB type.
FLAG	Transcodes or decodes character strings. **0**: encoding. **1**: decoding.

Example:

```
-- XML decoding
SELECT DBE_XMLGEN.CONVERT('<foo/>', 1);
 convert
---------
 <foo/>
(1 row)
-- XML encoding
SELECT DBE_XMLGEN.CONVERT('<foo><qwe</foo>', 0);
            convert
--------------------------------
 &lt;foo&gt;&lt;qwe&lt;/foo&gt;
(1 row)
```

- DBE_XMLGEN.NEWCONTEXT

  Initializes the common context handle.

  Function prototype:

  ```
  DBE_XMLGEN.NEWCONTEXT(QUERYSTRING IN VARCHAR2) RETURNS DBE_XMLGEN.CTXHANDLE;
  DBE_XMLGEN.NEWCONTEXT(QUERYSTRING IN SYS_REFCURSOR) RETURNS
  DBE_XMLGEN.CTXHANDLE;
  ```

  Parameter description

Table 10-477 DBE_XMLGEN.NEWCONTEXT parameters

Parameter	Description
QUERYSTRING	Queries SQL statement or SYS_REFCURSOR used to generate XML files.

Example:

```
-- Preset data.
CREATE TABLE IF NOT EXISTS department(department_id NUMBER, department_name
VARCHAR2(30), manager NUMBER, location NUMBER);
INSERT INTO department VALUES(10, 'administrator', 200, 1700);
INSERT INTO department VALUES(11, 'aaa', 200, 1700);
INSERT INTO department VALUES(12, 'bbb', 300, 1600);
INSERT INTO department VALUES(13, 'ccc', 400, 1600);
INSERT INTO department VALUES(14, 'aaa</row><a>asd</a><row>', 400, 1600);
INSERT INTO department VALUES(15, NULL, 500,1600);
INSERT INTO department VALUES(16, '!@#$ %^&*()+-=<>/\"a3_a', 400, 1600);
-- Initialize the common context handle.
DECLARE
    qryctx DBE_XMLGEN.CTXHANDLE;
    result CLOB;
BEGIN
 qryctx := DBE_XMLGEN.NEWCONTEXT('SELECT * FROM DEPARTMENT ORDER BY DEPARTMENT_ID');
 result:=DBE_XMLGEN.GETXML(qryctx);
 DBE_XMLGEN.CLOSECONTEXT(qryctx);
 DBE_OUTPUT.PUT_LINE(result);
END;
/

<?xml version="1.0"?>
<rowset>
<row>
  <department_id>10</department_id>
  <department_name>administrator</department_name>
  <manager>200</manager>
  <location>1700</location>
</row>
<row>
  <department_id>11</department_id>
  <department_name>aaa</department_name>
  <manager>200</manager>
  <location>1700</location>
</row>
<row>
  <department_id>12</department_id>
  <department_name>bbb</department_name>
  <manager>300</manager>
  <location>1600</location>
</row>
<row>
  <department_id>13</department_id>
  <department_name>ccc</department_name>
  <manager>400</manager>
  <location>1600</location>
</row>
<row>
  <department_id>14</department_id>
  <department_name>aaa&lt;/row&gt;&lt;a&gt;asd&lt;/a&gt;&lt;row&gt;</department_name>
  <manager>400</manager>
  <location>1600</location>
</row>
<row>
  <department_id>15</department_id>
```



```
  <manager>500</manager>
  <location>1600</location>
</row>
<row>
  <department_id>16</department_id>
  <department_name>!@#$%^&amp;amp;*()+-=&lt;&gt;/\"a3_a</department_name>
  <manager>400</manager>
  <location>1600</location>
</row>
</rowset>

-- Initialize the common context handle.
DECLARE
    lr SYS_REFCURSOR;
    qryctx DBE_XMLGEN.CTXHANDLE;
    result XMLTYPE;
BEGIN
    OPEN lr FOR SELECT department_id, department_name FROM DEPARTMENT ORDER BY
DEPARTMENT_ID;
    qryctx:=DBE_XMLGEN.NEWCONTEXT(lr);
 result:=DBE_XMLGEN.GETXMLTYPE(qryctx);
 DBE_XMLGEN.CLOSECONTEXT(qryctx);
 DBE_OUTPUT.PUT_LINE(result.getclobval);
END;
/
<?xml version="1.0"?>
<rowset>
<row>
  <department_id>10</department_id>
  <department_name>administrator</department_name>
</row>
<row>
  <department_id>11</department_id>
  <department_name>aaa</department_name>
</row>
<row>
  <department_id>12</department_id>
  <department_name>bbb</department_name>
</row>
<row>
  <department_id>13</department_id>
  <department_name>ccc</department_name>
</row>
<row>
  <department_id>14</department_id>
  <department_name>aaa&lt;/row&gt;&lt;a&gt;asd&lt;/a&gt;&lt;row&gt;</department_name>
</row>
<row>
  <department_id>15</department_id>
</row>
<row>
  <department_id>16</department_id>
  <department_name>!@#$%^&amp;amp;*()+-=&lt;&gt;/\"a3_a</department_name>
</row>
</rowset>
```

- DBE_XMLGEN.NEWCONTEXTFROMHIERARCHY

  Initializes the context handle with recursive elements.

  The data format must contain two columns. The first column is of the numeric type, and the second column is of the XML or XMLTYPE type. Generally, the value is generated by the connect by statement. The first column specifies the generation level.

  NOTE

  The number of nested layers in the generated XML file cannot exceed 50 million.

  Function prototype:


```
DBE_XMLGEN.NEWCONTEXTFROMHIERARCHY(QUERYSTRING IN VARCHAR2);
```

Parameter description

Table 10-478 DBE_XMLGEN.NEWCONTEXTFROMHIERARCHY parameters

Parameter	Description
QUERYSTRING	XML character string to be converted. The value is of the VARCHAR2 type.

Example:

```
-- Preset data.
CREATE TABLE tree_list(id NUMBER, name VARCHAR2(30), fid NUMBER);
INSERT INTO tree_list VALUES(1, 'a', NULL);
INSERT INTO tree_list VALUES(6, 'a-1-2', 2);
INSERT INTO tree_list VALUES(2, 'a-1', 1);
INSERT INTO tree_list VALUES(3, 'a-2', 1);
INSERT INTO tree_list VALUES(4, 'a-3', 1);
INSERT INTO tree_list VALUES(5, 'a-1-1', 2);
INSERT INTO tree_list VALUES(7, 'a-2-1', 3);
INSERT INTO tree_list VALUES(8, 'a-2-2', 3);
INSERT INTO tree_list VALUES(9, 'a-3-1', 4);
INSERT INTO tree_list VALUES(10, 'a-3-2', 4);
INSERT INTO tree_list VALUES(11, 'a-3-2-1', 10);
INSERT INTO tree_list VALUES(12, 'a-3-2-1-1', 11);
INSERT INTO tree_list VALUES(13, 'a-3-2-1-1-1', 12);
INSERT INTO tree_list VALUES(14, 'a-3-2-1-1-1-1', 13);
INSERT INTO tree_list VALUES(15, 'a-3-2-1-1-1-1-1', 14);
INSERT INTO tree_list VALUES(16, NULL, 14);
INSERT INTO tree_list VALUES(17, '<?q>', 14);
-- Recursive XML generation.
DECLARE
qryctx DBE_XMLGEN.CTXHANDLE;
result CLOB;
BEGIN
 qryctx := DBE_XMLGEN.NEWCONTEXTFROMHIERARCHY('SELECT level, xmlelement("children",
xmlelement("node_name", name)) ss from tree_list start with id=1 connect by prior id=fid');
 DBE_XMLGEN.USENULLATTRIBUTEINDICATOR(qryctx, true);
 result:=DBE_XMLGEN.GETXML(qryctx);
 DBE_XMLGEN.CLOSECONTEXT(qryctx);
 DBE_OUTPUT.PUT_LINE(result);
END;
/
<?xml version="1.0" encoding="utf-8"?>
<children>
 <node_name>a</node_name>
 <children>
  <node_name>a-3</node_name>
  <children>
   <node_name>a-3-2</node_name>
   <children>
    <node_name>a-3-2-1</node_name>
    <children>
     <node_name>a-3-2-1-1</node_name>
     <children>
      <node_name>a-3-2-1-1-1</node_name>
      <children>
       <node_name>a-3-2-1-1-1-1</node_name>
       <children>
        <node_name>&lt;?q&gt;</node_name>
       </children>
       <children>
        <node_name/>
```



```
          </children>
          <children>
            <node_name>a-3-2-1-1-1-1-1</node_name>
          </children>
        </children>
      </children>
    </children>
  </children>
  <children>
    <node_name>a-3-1</node_name>
  </children>
</children>
<children>
  <node_name>a-2</node_name>
  <children>
    <node_name>a-2-2</node_name>
  </children>
  <children>
    <node_name>a-2-1</node_name>
  </children>
</children>
<children>
  <node_name>a-1</node_name>
  <children>
    <node_name>a-1-1</node_name>
  </children>
  <children>
    <node_name>a-1-2</node_name>
  </children>
 </children>
</children>
```

- DBE_XMLGEN.SETCONVERTSPECIALCHARS

  Specifies whether the output XML file needs to be encoded. If XML encoding is canceled, XML injection may occur. If the XML is secured and the performance is considered, XML encoding is not required.

  Function prototype:

  ```
  DBE_XMLGEN.SETCONVERTSPECIALCHARS(CTX IN DBE_XMLGEN.CTXHANDLE, CONV IN BOOLEAN);
  ```

  Parameter description

  Table 10-479 DBE_XMLGEN.SETCONVERTSPECIALCHARS parameters

Parameter	Description
CTX	Context handle.
CONV	Specifies whether to encode the output XML file. • **true**: yes • **false**: no

  Example:

  ```
  -- Preset data.
  CREATE TABLE IF NOT EXISTS department(department_id NUMBER, department_name
  VARCHAR2(30), manager NUMBER, location NUMBER);
  INSERT INTO department VALUES(10, 'administrator', 200, 1700);
  INSERT INTO department VALUES(11, 'aaa', 200, 1700);
  INSERT INTO department VALUES(12, 'bbb', 300, 1600);
  INSERT INTO department VALUES(13, 'ccc', 400, 1600);
  ```



```
INSERT INTO department VALUES(14, 'aaa</row><a>asd</a><row>', 400, 1600);
INSERT INTO department VALUES(15, NULL, 500,1600);
INSERT INTO department VALUES(16, '!@#$ %^&*()+-=<>/\"a3_a', 400, 1600);
-- XML encoding.
DECLARE
    qryctx DBE_XMLGEN.CTXHANDLE;
    result CLOB;
BEGIN
 qryctx := DBE_XMLGEN.NEWCONTEXT('SELECT * from department where department_id=14');
 DBE_XMLGEN.SETCONVERTSPECIALCHARS(qryctx, true);
 result:=DBE_XMLGEN.GETXML(qryctx);
 DBE_XMLGEN.CLOSECONTEXT(qryctx);
 DBE_OUTPUT.PUT_LINE(result);
END;
/
<?xml version="1.0"?>
<rowset>
<row>
  <department_id>14</department_id>
  <department_name>aaa&lt;/row&gt;&lt;a&gt;asd&lt;/a&gt;&lt;row&gt;</department_name>
  <manager>400</manager>
  <location>1600</location>
</row>
</rowset>
-- No encoding is performed.
DECLARE
    qryctx DBE_XMLGEN.CTXHANDLE;
    result CLOB;
BEGIN
    qryctx := DBE_XMLGEN.NEWCONTEXT('SELECT * from department where department_id=14');
    DBE_XMLGEN.SETCONVERTSPECIALCHARS(qryctx, false);
    result:=DBE_XMLGEN.GETXML(qryctx);
    DBE_XMLGEN.CLOSECONTEXT(qryctx);
    DBE_OUTPUT.PUT_LINE(result);
END;
/
<?xml version="1.0"?>
<rowset>
<row>
  <department_id>14</department_id>
  <department_name>aaa</row><a>asd</a><row></department_name>
  <manager>400</manager>
  <location>1600</location>
</row>
</rowset>
-- Other non-XML special characters are not encoded.
DECLARE
    qryctx DBE_XMLGEN.CTXHANDLE;
    result CLOB;
BEGIN
    qryctx := DBE_XMLGEN.NEWCONTEXT('SELECT * from department where department_id=16');
    DBE_XMLGEN.SETCONVERTSPECIALCHARS(qryctx, true);
    result:=DBE_XMLGEN.GETXML(qryctx);
    DBE_XMLGEN.CLOSECONTEXT(qryctx);
    DBE_OUTPUT.PUT_LINE(result);
END;
/
<?xml version="1.0"?>
<rowset>
<row>
  <department_id>16</department_id>
  <department_name>!@#$%^&amp;*()+-=&lt;&gt;/\"a3_a</department_name>
  <manager>400</manager>
  <location>1600</location>
</row>
</rowset>
```

- DBE_XMLGEN.SETNULLHANDLING

  Sets how to display the null value in the XML file.

Function prototype:

```
DBE_XMLGEN.SETNULLHANDLING(CTX IN DBE_XMLGEN.CTXHANDLE, FLAG IN NUMBER := 0);
```

Parameter description

Table 10-480 DBE_XMLGEN.SETNULLHANDLING parameters

Parameter	Description
CTX	Context handle.
FLAG	Null value display format. **0**: The element is not displayed. **1**: The **xsi:nil="true"** attribute is added to the element. **2**: Self-closed elements are displayed.

Example:

```
-- Preset data.
CREATE TABLE IF NOT EXISTS department(department_id NUMBER, department_name
VARCHAR2(30), manager NUMBER, location NUMBER);
INSERT INTO department VALUES(10, 'administrator', 200, 1700);
INSERT INTO department VALUES(11, 'aaa', 200, 1700);
INSERT INTO department VALUES(12, 'bbb', 300, 1600);
INSERT INTO department VALUES(13, 'ccc', 400, 1600);
INSERT INTO department VALUES(14, 'aaa</row><a>asd</a><row>', 400, 1600);
INSERT INTO department VALUES(15, NULL, 500,1600);
INSERT INTO department VALUES(16, '!@#$ %^&*()+-=<>/\"a3_a', 400, 1600);

-- The default value of nullhandling is not used.
DECLARE
    result CLOB;
    qryctx DBE_XMLGEN.CTXHANDLE;
BEGIN
    qryctx:=DBE_XMLGEN.NEWCONTEXT('SELECT * from department where department_id=15');
    result:=DBE_XMLGEN.GETXML(qryctx);
    DBE_OUTPUT.PUT_LINE(result);
    DBE_XMLGEN.CLOSECONTEXT(qryctx);
END;
/
<?xml version="1.0"?>
<rowset>
<row>
  <department_id>15</department_id>
  <manager>500</manager>
  <location>1600</location>
</row>
</rowset>
-- nullhandling is 0.
DECLARE
    result CLOB;
    qryctx DBE_XMLGEN.CTXHANDLE;
BEGIN
    qryctx:=DBE_XMLGEN.NEWCONTEXT('SELECT * from department where department_id=15');
    DBE_XMLGEN.SETNULLHANDLING(qryctx, 0);
    result:=DBE_XMLGEN.GETXML(qryctx);
    DBE_OUTPUT.PUT_LINE(result);
    DBE_XMLGEN.CLOSECONTEXT(qryctx);
END;
/
```



```
<?xml version="1.0"?>
<rowset>
<row>
  <department_id>15</department_id>
  <manager>500</manager>
  <location>1600</location>
</row>
</rowset>
-- nullhandling 1
DECLARE
    result CLOB;
    qryctx DBE_XMLGEN.CTXHANDLE;
BEGIN
    qryctx:=DBE_XMLGEN.NEWCONTEXT('SELECT * from department where department_id=15');
    DBE_XMLGEN.SETNULLHANDLING(qryctx, 1);
    result:=DBE_XMLGEN.GETXML(qryctx);
    DBE_OUTPUT.PUT_LINE(result);
    DBE_XMLGEN.CLOSECONTEXT(qryctx);
END;
/
<?xml version="1.0"?>
<rowset xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<row>
  <department_id>15</department_id>
  <department_name xsi:nil="true"/>
  <manager>500</manager>
  <location>1600</location>
</row>
</rowset>
-- nullhandling 2
DECLARE
    result CLOB;
    qryctx DBE_XMLGEN.CTXHANDLE;
BEGIN
    qryctx:=DBE_XMLGEN.NEWCONTEXT('SELECT * from department where department_id=15');
    DBE_XMLGEN.SETNULLHANDLING(qryctx, 2);
    result:=DBE_XMLGEN.GETXML(qryctx);
    DBE_OUTPUT.PUT_LINE(result);
    DBE_XMLGEN.CLOSECONTEXT(qryctx);
END;
/
<?xml version="1.0"?>
<rowset>
<row>
  <department_id>15</department_id>
  <department_name/>
  <manager>500</manager>
  <location>1600</location>
</row>
</rowset>
```

- DBE_XMLGEN.SETROWSETTAG

  Sets the name of the XML root node.

  Function prototype:

  ```
  DBE_XMLGEN.SETROWSETTAG(CTX IN DBE_XMLGEN.CTXHANDLE, ROWSETTAGNAME IN VARCHAR2);
  ```

  Parameter description

  Table 10-481 DBE_XMLGEN.SETROWSETTAG parameters

Parameter	Description
CTX	Context handle.
ROWSETTAGNAME	Name of the XML root node.

Example:

```
-- Preset data.
CREATE TABLE IF NOT EXISTS department(department_id NUMBER, department_name
VARCHAR2(30), manager NUMBER, location NUMBER);
INSERT INTO department VALUES(10, 'administrator', 200, 1700);
INSERT INTO department VALUES(11, 'aaa', 200, 1700);
INSERT INTO department VALUES(12, 'bbb', 300, 1600);
INSERT INTO department VALUES(13, 'ccc', 400, 1600);
INSERT INTO department VALUES(14, 'aaa</row><a>asd</a><row>', 400, 1600);
INSERT INTO department VALUES(15, NULL, 500,1600);
INSERT INTO department VALUES(16, '!@#$ %^&*()+-=<>/\"a3_a', 400, 1600);
-- Set the root node name to asd.
DECLARE
    qryctx DBE_XMLGEN.CTXHANDLE;
    result CLOB;
BEGIN
    qryctx := DBE_XMLGEN.NEWCONTEXT('SELECT * FROM DEPARTMENT ORDER BY
DEPARTMENT_ID');
    DBE_XMLGEN.SETROWSETTAG(qryctx, 'asd');
    DBE_XMLGEN.SETROWTAG(qryctx, 'qwe');
    result:=DBE_XMLGEN.GETXML(qryctx);
    DBE_XMLGEN.CLOSECONTEXT(qryctx);
    DBE_OUTPUT.PUT_LINE(result);
END;
/
<?xml version="1.0"?>
<asd>
<qwe>
  <department_id>10</department_id>
  <department_name>administrator</department_name>
  <manager>200</manager>
  <location>1700</location>
</qwe>
<qwe>
  <department_id>11</department_id>
  <department_name>aaa</department_name>
  <manager>200</manager>
  <location>1700</location>
</qwe>
<qwe>
  <department_id>12</department_id>
  <department_name>bbb</department_name>
  <manager>300</manager>
  <location>1600</location>
</qwe>
<qwe>
  <department_id>13</department_id>
  <department_name>ccc</department_name>
  <manager>400</manager>
  <location>1600</location>
</qwe>
<qwe>
  <department_id>14</department_id>
  <department_name>aaa&lt;/row&gt;&lt;a&gt;asd&lt;/a&gt;&lt;row&gt;</department_name>
  <manager>400</manager>
  <location>1600</location>
</qwe>
<qwe>
  <department_id>15</department_id>
  <manager>500</manager>
  <location>1600</location>
</qwe>
<qwe>
  <department_id>16</department_id>
  <department_name>!@#$%^&amp;*()+-=&lt;&gt;/\"a3_a</department_name>
  <manager>400</manager>
  <location>1600</location>
</qwe>
</asd>
```


- DBE_XMLGEN.SETROWTAG

  Sets the tag name of each row of data in the XML file.

  Function prototype:

  ```
  DBE_XMLGEN.SETROWTAG(CTX IN DBE_XMLGEN.CTXHANDLE, ROWTAGNAME IN VARCHAR2);
  ```

  Parameter description

  Table 10-482 DBE_XMLGEN.SETROWTAG parameters

Parameter	Description
CTX	Context handle.
ROWTAGNAME	Tag name of each row of data.

  Example:

  ```
  -- Preset data.
  CREATE TABLE IF NOT EXISTS department(department_id NUMBER, department_name
  VARCHAR2(30), manager NUMBER, location NUMBER);
  INSERT INTO department VALUES(10, 'administrator', 200, 1700);
  INSERT INTO department VALUES(11, 'aaa', 200, 1700);
  INSERT INTO department VALUES(12, 'bbb', 300, 1600);
  INSERT INTO department VALUES(13, 'ccc', 400, 1600);
  INSERT INTO department VALUES(14, 'aaa</row><a>asd</a><row>', 400, 1600);
  INSERT INTO department VALUES(15, NULL, 500,1600);
  INSERT INTO department VALUES(16, '!@#$ %^&*()+-=<>/\"a3_a', 400, 1600);
  -- Set the tag name of each row of data to qwe.
  DECLARE
      qryctx DBE_XMLGEN.CTXHANDLE;
      result CLOB;
  BEGIN
      qryctx := DBE_XMLGEN.NEWCONTEXT('SELECT * FROM DEPARTMENT ORDER BY
  DEPARTMENT_ID');
      DBE_XMLGEN.SETROWSETTAG(qryctx, 'asd');
      DBE_XMLGEN.SETROWTAG(qryctx, 'qwe');
      result:=DBE_XMLGEN.GETXML(qryctx);
      DBE_XMLGEN.CLOSECONTEXT(qryctx);
      DBE_OUTPUT.PUT_LINE(result);
  END;
  /
  <?xml version="1.0"?>
  <asd>
  <qwe>
    <department_id>10</department_id>
    <department_name>administrator</department_name>
    <manager>200</manager>
    <location>1700</location>
  </qwe>
  <qwe>
    <department_id>11</department_id>
    <department_name>aaa</department_name>
    <manager>200</manager>
    <location>1700</location>
  </qwe>
  <qwe>
    <department_id>12</department_id>
    <department_name>bbb</department_name>
    <manager>300</manager>
    <location>1600</location>
  </qwe>
  <qwe>
    <department_id>13</department_id>
    <department_name>ccc</department_name>
  ```



```
 <manager>400</manager>
 <location>1600</location>
</qwe>
<qwe>
 <department_id>14</department_id>
 <department_name>aaa&lt;/row&gt;&lt;a&gt;asd&lt;/a&gt;&lt;row&gt;</department_name>
 <manager>400</manager>
 <location>1600</location>
</qwe>
<qwe>
 <department_id>15</department_id>
 <manager>500</manager>
 <location>1600</location>
</qwe>
<qwe>
 <department_id>16</department_id>
 <department_name>!@#$%^&amp;*()+-=&lt;&gt;/\"a3_a</department_name>
 <manager>400</manager>
 <location>1600</location>
</qwe>
</asd>
```

- DBE_XMLGEN.USENULLATTRIBUTEINDICATOR

  Adds the **xsi:nil="true"** attribute to the element where the **null** value is located in the XML file.

  Function prototype:

  ```
  DBE_XMLGEN.USENULLATTRIBUTEINDICATOR(CTX IN DBE_XMLGEN.CTXHANDLE, ATTRIND IN
  BOOLEAN);
  ```

  Parameter description

  Table 10-483 DBE_XMLGEN.USENULLATTRIBUTEINDICATOR parameters

Parameter	Description
CTX	Context handle.
ATTRIND	None.

  Example:

  ```
  -- Preset data.
  CREATE TABLE IF NOT EXISTS department(department_id NUMBER, department_name
  VARCHAR2(30), manager NUMBER, location NUMBER);
  INSERT INTO department VALUES(10, 'administrator', 200, 1700);
  INSERT INTO department VALUES(11, 'aaa', 200, 1700);
  INSERT INTO department VALUES(12, 'bbb', 300, 1600);
  INSERT INTO department VALUES(13, 'ccc', 400, 1600);
  INSERT INTO department VALUES(14, 'aaa</row><a>asd</a><row>', 400, 1600);
  INSERT INTO department VALUES(15, NULL, 500,1600);
  INSERT INTO department VALUES(16, '!@#$ %^&*()+-=<>/\"a3_a', 400, 1600);
  -- Add the xsi:nil="true" attribute to the null value.
  DECLARE
      result CLOB;
      qryctx DBE_XMLGEN.CTXHANDLE;
  BEGIN
      qryctx:=DBE_XMLGEN.NEWCONTEXT('SELECT * from department where department_id=15');
      DBE_XMLGEN.USENULLATTRIBUTEINDICATOR(qryctx, true);
      result:=DBE_XMLGEN.GETXML(qryctx);
      DBE_OUTPUT.PUT_LINE(result);
      DBE_XMLGEN.CLOSECONTEXT(qryctx);
  END;
  /
  <?xml version="1.0"?>
  ```



```
<rowset xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<row>
  <department_id>15</department_id>
  <department_name xsi:nil="true"/>
  <manager>500</manager>
  <location>1600</location>
</row>
</rowset>
```

- DBE_XMLGEN.USEITEMTAGSFORCOLL

  Adds the suffix **'_item'** to the element where the variable of the array type is located.

  Function prototype:

  ```
  DBE_XMLGEN.USEITEMTAGSFORCOLL(CTX IN DBE_XMLGEN.CTXHANDLE);
  ```

  Parameter description

  Table 10-484 DBE_XMLGEN.USEITEMTAGSFORCOLL parameters

Parameter	Description
CTX	Context handle.

  Example:

  ```
  -- Preset data.
  CREATE TABLE test_for_array(id INT[]);
  INSERT INTO test_for_array VALUES(ARRAY[1,2,3]);
  SELECT DBE_XMLGEN.GETXML('SELECT * from test_for_array');
  -- The suffix '_item' is added to the array type.
  DECLARE
  qryctx DBE_XMLGEN.CTXHANDLE;
  result CLOB;
  BEGIN
      qryctx := DBE_XMLGEN.NEWCONTEXT('SELECT * from test_for_array');
      DBE_XMLGEN.useItemTagsForColl(qryctx);
      result:=DBE_XMLGEN.GETXML(qryctx);
      DBE_XMLGEN.CLOSECONTEXT(qryctx);
      DBE_OUTPUT.PUT_LINE(result);
  END;
  /
  <?xml version="1.0"?>
  <rowset>
  <row>
    <id>
      <int4_ITEM>1</int4_ITEM>
      <int4_ITEM>2</int4_ITEM>
      <int4_ITEM>3</int4_ITEM>
    </id>
  </row>
  </rowset>
  ```

- DBE_XMLGEN.GETNUMROWSPROCESSED

  Views the number of data rows returned by getxml or getxmltype last time.

  Function prototype:

  ```
  DBE_XMLGEN.GETNUMROWSPROCESSED(CTX IN DBE_XMLGEN.CTXHANDLE);
  ```

  Parameter description

Table 10-485 DBE_XMLGEN.GETNUMROWSPROCESSED parameters

Parameter	Description
CTX	Context handle.

Example:

```
-- Preset data.
CREATE TABLE IF NOT EXISTS department(department_id NUMBER, department_name
VARCHAR2(30), manager NUMBER, location NUMBER);
INSERT INTO department VALUES(10, 'administrator', 200, 1700);
INSERT INTO department VALUES(11, 'aaa', 200, 1700);
INSERT INTO department VALUES(12, 'bbb', 300, 1600);
INSERT INTO department VALUES(13, 'ccc', 400, 1600);
INSERT INTO department VALUES(14, 'aaa</row><a>asd</a><row>', 400, 1600);
INSERT INTO department VALUES(15, NULL, 500,1600);
INSERT INTO department VALUES(16, '!@#$ %^&*()+-=<>/\"a3_a', 400, 1600);
-- getNumRowsProcessed
DECLARE
    qryctx DBE_XMLGEN.CTXHANDLE;
    result CLOB;
BEGIN
    qryctx:=DBE_XMLGEN.NEWCONTEXT('SELECT * from department');
    -- Maximum value returned for each query.
    DBE_XMLGEN.SETMAXROWS(qryctx, 4);
    LOOP
        result := DBE_XMLGEN.GETXML(qryctx);
        -- Number of records returned in this round of query.
        exit when DBE_XMLGEN.GETNUMROWSPROCESSED(qryctx) = 0;
        DBE_OUTPUT.PUT_LINE('-------'||DBE_XMLGEN.GETNUMROWSPROCESSED(qryctx)||'----------');
        DBE_OUTPUT.PUT_LINE(result);
        DBE_OUTPUT.PUT_LINE('********************');
    END LOOP;
    DBE_XMLGEN.CLOSECONTEXT(qryctx);
END;
/
-------4----------
<?xml version="1.0"?>
<rowset>
<row>
  <department_id>10</department_id>
  <department_name>administrator</department_name>
  <manager>200</manager>
  <location>1700</location>
</row>
<row>
  <department_id>11</department_id>
  <department_name>aaa</department_name>
  <manager>200</manager>
  <location>1700</location>
</row>
<row>
  <department_id>12</department_id>
  <department_name>bbb</department_name>
  <manager>300</manager>
  <location>1600</location>
</row>
<row>
  <department_id>13</department_id>
  <department_name>ccc</department_name>
  <manager>400</manager>
  <location>1600</location>
</row>
</rowset>

********************
```



```
-------3----------
<?xml version="1.0"?>
<rowset>
<row>
  <department_id>14</department_id>
  <department_name>aaa&lt;/row&gt;&lt;a&gt;asd&lt;/a&gt;&lt;row&gt;</department_name>
  <manager>400</manager>
  <location>1600</location>
</row>
<row>
  <department_id>15</department_id>
  <manager>500</manager>
  <location>1600</location>
</row>
<row>
  <department_id>16</department_id>
  <department_name>!@#$%^&amp;*()+-=&lt;&gt;/\"a3_a</department_name>
  <manager>400</manager>
  <location>1600</location>
</row>
</rowset>

********************
```

- DBE_XMLGEN.SETMAXROWS

  Sets the maximum number of rows returned by getxml.

  Function prototype:

  ```
  DBE_XMLGEN.SETMAXROWS(CTX IN DBE_XMLGEN.CTXHANDLE, MAXROWS IN NUMBER);
  ```

  Parameter description

  Table 10-486 DBE_XMLGEN.SETMAXROWS parameters

Parameter	Description
CTX	Context handle.
MAXROWS	Maximum number of rows returned by each getxml operation.

  Example:

  ```
  -- Preset data.
  CREATE TABLE IF NOT EXISTS department(department_id NUMBER, department_name
  VARCHAR2(30), manager NUMBER, location NUMBER);
  INSERT INTO department VALUES(10, 'administrator', 200, 1700);
  INSERT INTO department VALUES(11, 'aaa', 200, 1700);
  INSERT INTO department VALUES(12, 'bbb', 300, 1600);
  INSERT INTO department VALUES(13, 'ccc', 400, 1600);
  INSERT INTO department VALUES(14, 'aaa</row><a>asd</a><row>', 400, 1600);
  INSERT INTO department VALUES(15, NULL, 500,1600);
  INSERT INTO department VALUES(16, '!@#$ %^&*()+-=<>/\"a3_a', 400, 1600);
  -- getNumRowsProcessed
  DECLARE
      qryctx DBE_XMLGEN.CTXHANDLE;
   result CLOB;
  BEGIN
   qryctx:=DBE_XMLGEN.NEWCONTEXT('SELECT * from department');
      -- Maximum value returned for each query.
   DBE_XMLGEN.SETMAXROWS(qryctx, 4);
   LOOP
          result := DBE_XMLGEN.GETXML(qryctx);
          -- Number of records returned in this round of query.
    exit when DBE_XMLGEN.GETNUMROWSPROCESSED(qryctx) = 0;
  ```



```
 DBE_OUTPUT.PUT_LINE('-------'||DBE_XMLGEN.GETNUMROWSPROCESSED(qryctx)||'----------');
 DBE_OUTPUT.PUT_LINE(result);
 DBE_OUTPUT.PUT_LINE('********************');
   END LOOP;
 DBE_XMLGEN.CLOSECONTEXT(qryctx);
END;
/
-------4----------
<?xml version="1.0"?>
<rowset>
<row>
 <department_id>10</department_id>
 <department_name>administrator</department_name>
 <manager>200</manager>
 <location>1700</location>
</row>
<row>
 <department_id>11</department_id>
 <department_name>aaa</department_name>
 <manager>200</manager>
 <location>1700</location>
</row>
<row>
 <department_id>12</department_id>
 <department_name>bbb</department_name>
 <manager>300</manager>
 <location>1600</location>
</row>
<row>
 <department_id>13</department_id>
 <department_name>ccc</department_name>
 <manager>400</manager>
 <location>1600</location>
</row>
</rowset>

********************
-------3----------
<?xml version="1.0"?>
<rowset>
<row>
 <department_id>14</department_id>
 <department_name>aaa&lt;/row&gt;&lt;a&gt;asd&lt;/a&gt;&lt;row&gt;</department_name>
 <manager>400</manager>
 <location>1600</location>
</row>
<row>
 <department_id>15</department_id>
 <manager>500</manager>
 <location>1600</location>
</row>
<row>
 <department_id>16</department_id>
 <department_name>!@#$%^&amp;*()+-=&lt;&gt;/\"a3_a</department_name>
 <manager>400</manager>
 <location>1600</location>
</row>
</rowset>

********************
```

- DBE_XMLGEN.SETSKIPROWS

  Sets the number of SQL rows to be skipped.

  Function prototype:

  ```
  DBE_XMLGEN.SETSKIPROWS(CTX IN DBE_XMLGEN.CTXHANDLE, SKIPROWS IN NUMBER);
  ```

  Parameter description

Table 10-487 DBE_XMLGEN.SETSKIPROWS parameters

Parameter	Description
CTX	Context handle.
SKIPROWS	Number of SQL header rows that are skipped.

Example:

```
CREATE TABLE IF NOT EXISTS department(department_id NUMBER, department_name
VARCHAR2(30), manager NUMBER, location NUMBER);
INSERT INTO department VALUES(10, 'administrator', 200, 1700);
INSERT INTO department VALUES(11, 'aaa', 200, 1700);
INSERT INTO department VALUES(12, 'bbb', 300, 1600);
INSERT INTO department VALUES(13, 'ccc', 400, 1600);
INSERT INTO department VALUES(14, 'aaa</row><a>asd</a><row>', 400, 1600);
INSERT INTO department VALUES(15, NULL, 500,1600);
INSERT INTO department VALUES(16, '!@#$ %^&*()+-=<>/\"a3_a', 400, 1600);
-- Use setskiprows to skip five rows.
declare
    result CLOB;
    qryctx DBE_XMLGEN.CTXHANDLE;
BEGIN
 qryctx:=DBE_XMLGEN.NEWCONTEXT('SELECT * from department');
 DBE_XMLGEN.SETSKIPROWS(qryctx, 5);
 result:=DBE_XMLGEN.GETXML(qryctx);
 DBE_OUTPUT.PUT_LINE(result);
 DBE_XMLGEN.CLOSECONTEXT(qryctx);
END;
/
<?xml version="1.0"?>
<rowset>
<row>
  <department_id>15</department_id>
  <manager>500</manager>
  <location>1600</location>
</row>
<row>
  <department_id>16</department_id>
  <department_name>!@#$%^&amp;*()+-=&lt;&gt;/\"a3_a</department_name>
  <manager>400</manager>
  <location>1600</location>
</row>
```

- DBE_XMLGEN.RESTARTQUERY

  Restarts the SQL.

  Function prototype:

  ```
  DBE_XMLGEN.RESTARTQUERY(CTX IN DBE_XMLGEN.CTXHANDLE);
  ```

  Parameter description

  Table 10-488 DBE_XMLGEN.RESTARTQUERY parameters

Parameter	Description
CTX	Context handle.

  Example:


```
CREATE TABLE IF NOT EXISTS department(department_id NUMBER, department_name
VARCHAR2(30), manager NUMBER, location NUMBER);
INSERT INTO department VALUES(10, 'administrator', 200, 1700);
INSERT INTO department VALUES(11, 'aaa', 200, 1700);
INSERT INTO department VALUES(12, 'bbb', 300, 1600);
INSERT INTO department VALUES(13, 'ccc', 400, 1600);
INSERT INTO department VALUES(14, 'aaa</row><a>asd</a><row>', 400, 1600);
INSERT INTO department VALUES(15, NULL, 500,1600);
INSERT INTO department VALUES(16, '!@#$ %^&*()+-=<>/\"a3_a', 400, 1600);

DECLARE
    qryctx DBE_XMLGEN.CTXHANDLE;
 result CLOB;
BEGIN
 qryctx:=DBE_XMLGEN.NEWCONTEXT('SELECT * from department');
    -- Maximum value returned for each query.
 DBE_XMLGEN.SETMAXROWS(qryctx, 4);
 LOOP
        result := DBE_XMLGEN.GETXML(qryctx);
        -- Number of records returned in this round of query.
        exit when DBE_XMLGEN.GETNUMROWSPROCESSED(qryctx) = 0;
        DBE_OUTPUT.PUT_LINE('-------'||DBE_XMLGEN.GETNUMROWSPROCESSED(qryctx)||'----------');
        DBE_OUTPUT.PUT_LINE(result);
        DBE_OUTPUT.PUT_LINE('********************');
    END LOOP;
    -- Retry the query.
    DBE_XMLGEN.RESTARTQUERY(qryctx);
    result := DBE_XMLGEN.GETXML(qryctx);
    DBE_OUTPUT.PUT_LINE('-------'||DBE_XMLGEN.GETNUMROWSPROCESSED(qryctx)||'----------');
    DBE_OUTPUT.PUT_LINE(result);
    DBE_OUTPUT.PUT_LINE('********************');
    DBE_XMLGEN.CLOSECONTEXT(qryctx);
END;
/
--------4----------
<?xml version="1.0"?>
<rowset>
<row>
  <department_id>10</department_id>
  <department_name>administrator</department_name>
  <manager>200</manager>
  <location>1700</location>
</row>
<row>
  <department_id>11</department_id>
  <department_name>aaa</department_name>
  <manager>200</manager>
  <location>1700</location>
</row>
<row>
  <department_id>12</department_id>
  <department_name>bbb</department_name>
  <manager>300</manager>
  <location>1600</location>
</row>
<row>
  <department_id>13</department_id>
  <department_name>ccc</department_name>
  <manager>400</manager>
  <location>1600</location>
</row>
</rowset>

********************
--------3----------
<?xml version="1.0"?>
<rowset>
<row>
  <department_id>14</department_id>
```



```
 <department_name>aaa&lt;/row&gt;&lt;a&gt;asd&lt;/a&gt;&lt;row&gt;</department_name>
 <manager>400</manager>
 <location>1600</location>
</row>
<row>
 <department_id>15</department_id>
 <manager>500</manager>
 <location>1600</location>
</row>
<row>
 <department_id>16</department_id>
 <department_name>!@#$%^&amp;*()+-=&lt;&gt;/\"a3_a</department_name>
 <manager>400</manager>
 <location>1600</location>
</row>
</rowset>

********************
-------4----------
<?xml version="1.0"?>
<rowset>
<row>
 <department_id>10</department_id>
 <department_name>administrator</department_name>
 <manager>200</manager>
 <location>1700</location>
</row>
<row>
 <department_id>11</department_id>
 <department_name>aaa</department_name>
 <manager>200</manager>
 <location>1700</location>
</row>
<row>
 <department_id>12</department_id>
 <department_name>bbb</department_name>
 <manager>300</manager>
 <location>1600</location>
</row>
<row>
 <department_id>13</department_id>
 <department_name>ccc</department_name>
 <manager>400</manager>
 <location>1600</location>
</row>
</rowset>

********************
```

- DBE_XMLGEN.GETXMLTYPE

  Returns the XML text of the XMLTYPE type.

  Function prototype:

  ```
  DBE_XMLGEN.GETXMLTYPE(SQLQUERY IN VARCHAR2, DTDORSCHEMA IN NUMBER := 0) RETURNS XMLTYPE;
  DBE_XMLGEN.GETXMLTYPE(CTX IN DBE_XMLGEN.CTXHANDLE, DTDORSCHEMA IN NUMBER := 0) RETURNS XMLTYPE;
  ```

  Parameter description

  Table 10-489 DBE_XMLGEN.GETXMLTYP parameters

Parameter	Description
SQLQUERY	Query SQL statements that need to be converted into XML files.

Parameter	Description
DTDORSCHEMA	None.
CTX	Context handle.

Example:

```
CREATE TABLE IF NOT EXISTS department(department_id NUMBER, department_name
VARCHAR2(30), manager NUMBER, location NUMBER);
INSERT INTO department VALUES(10, 'administrator', 200, 1700);
INSERT INTO department VALUES(11, 'aaa', 200, 1700);
INSERT INTO department VALUES(12, 'bbb', 300, 1600);
INSERT INTO department VALUES(13, 'ccc', 400, 1600);
INSERT INTO department VALUES(14, 'aaa</row><a>asd</a><row>', 400, 1600);
INSERT INTO department VALUES(15, NULL, 500,1600);
INSERT INTO department VALUES(16, '!@#$ %^&*()+-=<>/\"a3_a', 400, 1600);
DECLARE
    lr SYS_REFCURSOR;
    qryctx DBE_XMLGEN.CTXHANDLE;
    result XMLTYPE;
BEGIN
    OPEN lr FOR SELECT department_id, department_name FROM DEPARTMENT ORDER BY
DEPARTMENT_ID;
    qryctx:=DBE_XMLGEN.NEWCONTEXT(lr);
 result:=DBE_XMLGEN.GETXMLTYPE(qryctx);
 DBE_XMLGEN.CLOSECONTEXT(qryctx);
       -- The returned value is of the XMLTYPE type. Therefore, the value can be output by put_line
only after being converted by getclobval.
 DBE_OUTPUT.PUT_LINE(result.getclobval);
END;
/
<?xml version="1.0"?>
<rowset>
<row>
  <department_id>10</department_id>
  <department_name>administrator</department_name>
</row>
<row>
  <department_id>11</department_id>
  <department_name>aaa</department_name>
</row>
<row>
  <department_id>12</department_id>
  <department_name>bbb</department_name>
</row>
<row>
  <department_id>13</department_id>
  <department_name>ccc</department_name>
</row>
<row>
  <department_id>14</department_id>
  <department_name>aaa&lt;/row&gt;&lt;a&gt;asd&lt;/a&gt;&lt;row&gt;</department_name>
</row>
<row>
  <department_id>15</department_id>
</row>
<row>
  <department_id>16</department_id>
  <department_name>!@#$%^&amp;*()+-=&lt;&gt;/\"a3_a</department_name>
</row>
</rowset>
```

- DBE_XMLGEN.GETXML

  Returns the XML text of the CLOB type.

Function prototype:

```
DBE_XMLGEN.GETXML(SQLQUERY IN VARCHAR2, DTDORSCHEMA IN NUMBER := 0) RETURNS CLOB;
DBE_XMLGEN.GETXML(CTX IN DBE_XMLGEN.CTXHANDLE, DTDORSCHEMA IN NUMBER := 0)
RETURNS CLOB;
DBE_XMLGEN.GETXML(CTX IN DBE_XMLGEN.CTXHANDLE, TMPCLOB INOUT CLOB, DTDORSCHEMA
IN NUMBER := 0);
```

Parameter description

Table 10-490 DBE_XMLGEN.GETXML parameters

Parameter	Description
SQLQUERY	Query SQL statements that need to be converted into XML files.
DTDORSCHEMA	None.
CTX	Context handle.
TMPCLOB	CLOB variable for storing the output XML.

Example:

```
CREATE TABLE IF NOT EXISTS department(department_id NUMBER, department_name
VARCHAR2(30), manager NUMBER, location NUMBER);
INSERT INTO department VALUES(10, 'administrator', 200, 1700);
INSERT INTO department VALUES(11, 'aaa', 200, 1700);
INSERT INTO department VALUES(12, 'bbb', 300, 1600);
INSERT INTO department VALUES(13, 'ccc', 400, 1600);
INSERT INTO department VALUES(14, 'aaa</row><a>asd</a><row>', 400, 1600);
INSERT INTO department VALUES(15, NULL, 500,1600);
INSERT INTO department VALUES(16, '!@#$ %^&*()+-=<>/\"a3_a', 400, 1600);
DECLARE
    qryctx DBE_XMLGEN.CTXHANDLE;
 result CLOB;
BEGIN
 qryctx:=DBE_XMLGEN.NEWCONTEXT('SELECT * from department');
    -- Maximum value returned for each query.
 DBE_XMLGEN.SETMAXROWS(qryctx, 4);
 LOOP
        result := DBE_XMLGEN.GETXML(qryctx);
        -- Number of records returned in this round of query.
        exit when DBE_XMLGEN.GETNUMROWSPROCESSED(qryctx) = 0;
        DBE_OUTPUT.PUT_LINE('-------'||DBE_XMLGEN.GETNUMROWSPROCESSED(qryctx)||'----------');
        DBE_OUTPUT.PUT_LINE(result);
        DBE_OUTPUT.PUT_LINE('********************');
    END LOOP;
    DBE_XMLGEN.RESTARTQUERY(qryctx);
    result := DBE_XMLGEN.GETXML(qryctx);
    DBE_OUTPUT.PUT_LINE('-------'||DBE_XMLGEN.GETNUMROWSPROCESSED(qryctx)||'----------');
    DBE_OUTPUT.PUT_LINE(result);
    DBE_OUTPUT.PUT_LINE('********************');
    DBE_XMLGEN.CLOSECONTEXT(qryctx);
END;
/
-------4----------
<?xml version="1.0"?>
<rowset>
<row>
  <department_id>10</department_id>
  <department_name>administrator</department_name>
  <manager>200</manager>
```



```
  <location>1700</location>
</row>
<row>
  <department_id>11</department_id>
  <department_name>aaa</department_name>
  <manager>200</manager>
  <location>1700</location>
</row>
<row>
  <department_id>12</department_id>
  <department_name>bbb</department_name>
  <manager>300</manager>
  <location>1600</location>
</row>
<row>
  <department_id>13</department_id>
  <department_name>ccc</department_name>
  <manager>400</manager>
  <location>1600</location>
</row>
</rowset>

********************
-------3----------
<?xml version="1.0"?>
<rowset>
<row>
  <department_id>14</department_id>
  <department_name>aaa&lt;/row&gt;&lt;a&gt;asd&lt;/a&gt;&lt;row&gt;</department_name>
  <manager>400</manager>
  <location>1600</location>
</row>
<row>
  <department_id>15</department_id>
  <manager>500</manager>
  <location>1600</location>
</row>
<row>
  <department_id>16</department_id>
  <department_name>!@#$%^&amp;*()+-=&lt;&gt;/\"a3_a</department_name>
  <manager>400</manager>
  <location>1600</location>
</row>
</rowset>

********************
-------4----------
<?xml version="1.0"?>
<rowset>
<row>
  <department_id>10</department_id>
  <department_name>administrator</department_name>
  <manager>200</manager>
  <location>1700</location>
</row>
<row>
  <department_id>11</department_id>
  <department_name>aaa</department_name>
  <manager>200</manager>
  <location>1700</location>
</row>
<row>
  <department_id>12</department_id>
  <department_name>bbb</department_name>
  <manager>300</manager>
  <location>1600</location>
</row>
<row>
  <department_id>13</department_id>
```



```
 <department_name>ccc</department_name>
 <manager>400</manager>
 <location>1600</location>
</row>
</rowset>

********************
```

- DBE_XMLGEN.CLOSECONTEXT

  Disables the context handle.

  Function prototype:

  ```
  DBE_XMLGEN.CLOSECONTEXT(CTX IN DBE_XMLGEN.CTXHANDLE);
  ```

  Parameter description

  Table 10-491 DBE_XMLGEN.CLOSECONTEXT parameters

Parameter	Description
CTX	Context handle.

  Example:

```
CREATE TABLE IF NOT EXISTS department(department_id NUMBER, department_name
VARCHAR2(30), manager NUMBER, location NUMBER);
INSERT INTO department VALUES(10, 'administrator', 200, 1700);
INSERT INTO department VALUES(11, 'aaa', 200, 1700);
INSERT INTO department VALUES(12, 'bbb', 300, 1600);
INSERT INTO department VALUES(13, 'ccc', 400, 1600);
INSERT INTO department VALUES(14, 'aaa</row><a>asd</a><row>', 400, 1600);
INSERT INTO department VALUES(15, NULL, 500,1600);
INSERT INTO department VALUES(16, '!@#$ %^&*()+-=<>/\"a3_a', 400, 1600);
-- Disable the context.
DECLARE
    qryctx DBE_XMLGEN.CTXHANDLE;
    result CLOB;
BEGIN
 qryctx := DBE_XMLGEN.NEWCONTEXT('SELECT * from department where department_id=14');
 DBE_XMLGEN.SETCONVERTSPECIALCHARS(qryctx, false);
 result:=DBE_XMLGEN.GETXML(qryctx);
 DBE_XMLGEN.CLOSECONTEXT(qryctx);
 DBE_OUTPUT.PUT_LINE(result);
END;
/
<?xml version="1.0"?>
<rowset>
<row>
 <department_id>14</department_id>
 <department_name>aaa</row><a>asd</a><row></department_name>
 <manager>400</manager>
 <location>1600</location>
</row>
</rowset>
```

10.13 Retry Management

Retry is a process in which the database executes an SQL statement or stored procedure (including anonymous block) again in the case of execution failure, improving the execution success rate and user experience. The database checks the error code and retry configuration to determine whether to retry.

- If the execution fails, the system rolls back the executed statements and executes the stored procedure again.

  Example:

```
gaussdb=# CREATE OR REPLACE PROCEDURE retry_basic ( IN  x INT)
AS
BEGIN
     INSERT INTO t1 (a) VALUES (x);
     INSERT INTO t1 (a) VALUES (x+1);
END;
/
CREATE PROCEDURE
gaussdb=# CALL retry_basic(1);
ERROR:  relation "t1" does not exist on datanode
LINE 1: INSERT INTO t1 (a) VALUES (x)
                    ^
QUERY:  INSERT INTO t1 (a) VALUES (x)
CONTEXT:  PL/pgSQL function retry_basic(integer) line 3 at SQL statement
```

10.14 Debugging

Syntax

RAISE

The syntax of RAISE is as follows:

Figure 10-36 raise_format::=

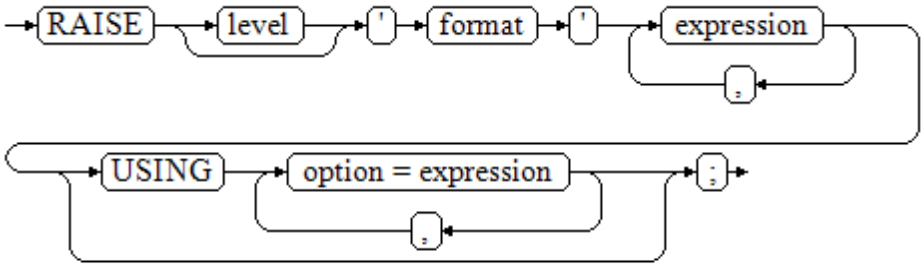


Figure 10-37 raise_condition::=

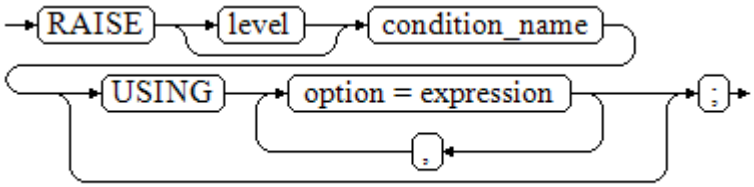


Figure 10-38 raise_sqlstate::=

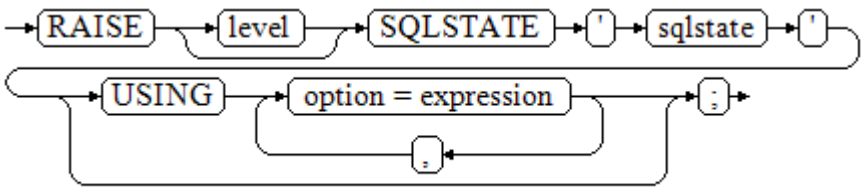


Figure 10-39 raise_option::=

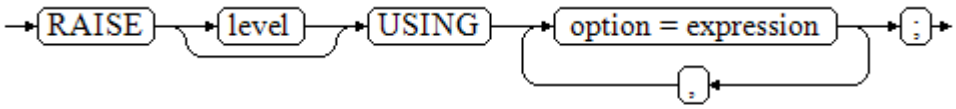
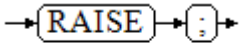


Figure 10-40 raise::=



Parameter description:

- The **level** option is used to specify the error level, that is, **DEBUG**, **LOG**, **INFO**, **NOTICE**, **WARNING**, or **EXCEPTION** (default). **EXCEPTION** throws an error that normally terminates the current transaction and the others only generate information at their levels. The GUC parameters **log_min_messages** and **client_min_messages** determine whether the error messages of specific levels are reported to the client and are written to the server log.
- **format**: specifies the error message text to be reported, a format string. The format string can be appended with an expression for insertion to the message text. In a format string, **%** is replaced by the parameter value attached to format and **%%** is used to print **%**. For example:

```
--v_job_id replaces % in the string.
RAISE NOTICE 'Calling cs_create_job(%)',v_job_id;
```

- **option = expression**: inserts additional information to an error report. The keyword option can be **MESSAGE**, **DETAIL**, **HINT**, or **ERRCODE**, and each expression can be any string.
  - **MESSAGE**: specifies the error message text. This option cannot be used in a **RAISE** statement that contains a format character string in front of **USING**.
  - **DETAIL**: specifies detailed information of an error.
  - **HINT**: prints hint information.
  - **ERRCODE**: designates an error code (SQLSTATE) to a report. A condition name or a five-character SQLSTATE error code can be used.
- **condition_name**: specifies the condition name corresponding to the error code.
- **sqlstate**: specifies the error code.

If neither a condition name nor an **SQLSTATE** is designated in a **RAISE EXCEPTION** command, the **RAISE EXCEPTION (P0001)** is used by default. If no message text is designated, the condition name or SQLSTATE is used as the message text by default.

NOTICE

- If the **SQLSTATE** designates an error code, the error code is not limited to a defined error code. It can be any error code containing five digits or ASCII uppercase rather than **00000**. Do not use an error code ended with three zeros because such error codes are category codes and can be captured by the whole category.
- In O-compatible mode, SQLCODE is equivalent to SQLSTATE.

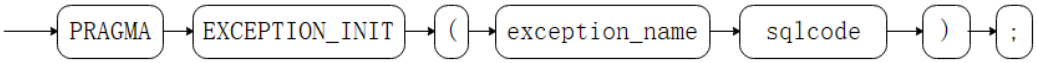
NOTE

The syntax described in **Figure 10-40** does not append any parameter. This form is used only for the **EXCEPTION** statement in a **BEGIN** block so that the error can be re-processed.

EXCEPTION_INIT

In O-compatible mode, EXCEPTION_INIT can be used to define the SQLCODE error code. The syntax is as follows:

Figure 10-41 exception_init::=



Parameter description:

- **exception_name** indicates the name of the exception declared by the user. The EXCEPTION_INIT syntax must follow the declared exception.
- **sqlcode** is a customized SQL code, which must be a negative integer ranging from –2147483647 to –1.

Precautions:

- This function can be used in A-compatible mode but cannot be bound to system error codes.
- Only exception_init can be used to assign a value to sqlcode joined with exception. Other value assignment modes are invalid.
- Cross-database.schema.package.exception calling is not supported.

NOTICE

When **EXCEPTION_INIT** is used to customize an SQL code, SQLSTATE is equivalent to SQLCODE, and SQLERRM is in the format of *xxx*: **non-GaussDB Exception**. For example, if the customized SQL code is **–1**, SQLSTATE is **–1** and SQLERRM is **1: non-GaussDB Exception**.

Examples

Display error and hint information when a transaction terminates:

```
CREATE OR REPLACE PROCEDURE proc_raise1(user_id in integer)
AS
```

```
BEGIN
RAISE EXCEPTION 'Noexistence ID --> %',user_id USING HINT = 'Please check your user ID';
END;
/

call proc_raise1(300011);

-- Execution result:
ERROR:  Noexistence ID --> 300011
HINT:  Please check your user ID
```

Two methods are available for setting **SQLSTATE**:

```
CREATE OR REPLACE PROCEDURE proc_raise2(user_id in integer)
AS
BEGIN
RAISE 'Duplicate user ID: %',user_id USING ERRCODE = 'unique_violation';
END;
/

\set VERBOSITY verbose
call proc_raise2(300011);

-- Execution result:
ERROR:  Duplicate user ID: 300011
SQLSTATE: 23505
LOCATION:  exec_stmt_raise, pl_exec.cpp:3482
```

If the main parameter is a condition name or **SQLSTATE**, the following applies:

RAISE division_by_zero;

RAISE SQLSTATE '22012';

For example:

```
CREATE OR REPLACE PROCEDURE division(div in integer, dividend in integer)
AS
DECLARE
res int;
    BEGIN
    IF dividend=0 THEN
        RAISE division_by_zero;
        RETURN;
    ELSE
        res := div/dividend;
        RAISE INFO 'division result: %', res;
        RETURN;
    END IF;
    END;
/
call division(3,0);

-- Execution result:
ERROR:  division_by_zero
```

Alternatively:

```
RAISE unique_violation USING MESSAGE = 'Duplicate user ID: ' || user_id;
```

In O-compatible mode, EXCEPTION_INIT can be used to customize error codes SQLCODE.

```
declare
    deadlock_detected exception;
    pragma exception_init(deadlock_detected, -1);
begin
    if 1 > 0 then
        raise deadlock_detected;
    end if;
exception
```



```
    when deadlock_detected then
        raise notice 'sqlcode:%,sqlstate:%,sqlerrm:%',sqlcode,sqlstate,sqlerrm;
end;
/
-- Execution result:
NOTICE:  sqlcode:-1,sqlstate:-1,sqlerrm: 1: non-GaussDB Exception
```

10.15 Package

A package is a combination of PL/SQL programs, such as stored procedures, functions, variables, constants, and cursors. It is object-oriented and can encapsulate PL/SQL program design elements. Functions in a package are created, deleted, and modified in a unified manner.

A package contains two parts: package specifications and package body. The declaration contained in the package specifications can be accessed by external functions and anonymous blocks. The declaration contained in the package body cannot be accessed by external functions or anonymous blocks, but can be accessed only by functions and stored procedures in the package body.

For details about how to create a package, see **CREATE PACKAGE**.

NOTICE

- Cross-package variables cannot be used as control variables in the for loops.
- Types defined in a package cannot be deleted or modified, and cannot be used to define tables.
- Cursor variables cannot be referenced in SCHEMA.PACKAGE.CURSOR mode.
- A cursor with parameters can be opened only in the current package.
- A package variable cannot be used as the default value of a function or stored procedure parameter.

11 Autonomous Transaction

An autonomous transaction is an independent transaction that is started during the execution of a primary transaction. Committing and rolling back an autonomous transaction does not affect the data that has been committed by the primary transaction. In addition, an autonomous transaction is not affected by the primary transaction.

Autonomous transactions are defined in stored procedures, functions, anonymous blocks, and packages, and are declared using the **PRAGMA AUTONOMOUS_TRANSACTION** keyword.

11.1 Restrictions

CAUTION

- When an autonomous transaction is executed, an autonomous transaction session is started in the background. You can use **max_concurrent_autonomous_transactions** to set the maximum number of concurrent autonomous transactions. The value range is 0 to 10000, and the default value is **10**.
- When **max_concurrent_autonomous_transactions** is set to **0**, autonomous transactions cannot be executed.
- After a new session is started for an autonomous transaction, the default session parameters are used and objects (including session-level variables, local temporary variables, and global temporary table data) of the primary session are not shared.
- Theoretically, the upper limit of autonomous transactions is 10000. Actually, the upper limit is a dynamic value. For details, see the description of the GUC parameter **max_concurrent_autonomous_transactions**.
- Autonomous transactions are affected by the communication buffer. The size of the information returned to the client is limited by the length of the communication buffer. If the size exceeds the length of the communication buffer, an error is reported.

- A trigger function does not support autonomous transactions.

```
gaussdb=# CREATE TABLE test_trigger_des_tbl(id1 int, id2 int, id3 int);
CREATE TABLE

gaussdb=# CREATE OR REPLACE FUNCTION tri_insert_func() RETURNS TRIGGER AS
$$
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
INSERT INTO test_trigger_des_tbl VALUES(new.id1, new.id2, new.id3);
RETURN new;
END$$ LANGUAGE plpgsql;
ERROR:  Triggers do not support autonomous transactions
DETAIL:  N/A

gaussdb=# DROP TABLE test_trigger_des_tbl;
DROP TABLE
```

- Autonomous transactions cannot be called by non-top-layer anonymous blocks (but can only be called by top-layer autonomous transactions, including stored procedures, functions, and anonymous blocks).

```
gaussdb=# CREATE TABLE t1(a INT ,b TEXT);
CREATE TABLE

gaussdb=# DECLARE
--PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
DBE_OUTPUT.PRINT_LINE('JUST USE CALL.');
INSERT INTO t1 VALUES(1,'CAN YOU ROLLBACK!');
END;
INSERT INTO t1 VALUES(2,'I WILL ROLLBACK!');
ROLLBACK;
END;
/
JUST USE CALL.
ANONYMOUS BLOCK EXECUTE
gaussdb=# SELECT * FROM t1;
 a | b
---+---
(0 rows)

gaussdb=# DROP TABLE t1;
DROP TABLE
```

- In an autonomous transaction, the **ref cursor** parameter can be passed only through the **PROCEDURE OUT** parameter. The **ref cursor** parameter cannot be passed through **IN**, **INOUT**, or **FUNCTION**. In a stored procedure, the **ref cursor** parameter can be passed only through the **OUT** parameter of an autonomous transaction. The **ref cursor** parameter cannot be directly called on the client (such as gsql and JDBC).

```
-- Create a table.
gaussdb=# CREATE TABLE sections(section_id INT);
CREATE TABLE
gaussdb=# INSERT INTO sections VALUES(1);
INSERT 0 1
gaussdb=# INSERT INTO sections VALUES(1);
INSERT 0 1
gaussdb=# INSERT INTO sections VALUES(1);
INSERT 0 1
gaussdb=# INSERT INTO sections VALUES(1);
INSERT 0 1
```

  a. The **PROCEDURE OUT** output parameter passes the **ref cursor** parameter (supported).


```
gaussdb=# CREATE OR REPLACE PROCEDURE proc_sys_ref(OUT c1 REFCURSOR)
IS
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
 OPEN c1 FOR SELECT section_id FROM sections ORDER BY section_id;

END;
/
CREATE PROCEDURE

gaussdb=# CREATE OR REPLACE PROCEDURE proc_sys_call() AS
DECLARE
  c1 SYS_REFCURSOR;
  temp NUMBER(4);
BEGIN
  proc_sys_ref(c1);
  IF c1%ISOPEN THEN
    RAISE NOTICE '%','OK';
 END IF;

  LOOP
    FETCH c1 INTO temp;
    RAISE NOTICE '%',c1%ROWCOUNT;
    EXIT WHEN c1%NOTFOUND;
  END LOOP;
END;
/
CREATE PROCEDURE

gaussdb=# CALL proc_sys_call();
NOTICE:  OK
NOTICE:  1
NOTICE:  2
NOTICE:  3
NOTICE:  4
NOTICE:  4
 proc_sys_call
---------------

(1 row)
gaussdb=# DROP PROCEDURE proc_sys_ref;
DROP PROCEDURE
gaussdb=# DROP PROCEDURE proc_sys_call;
DROP PROCEDURE
```

b. The **PROCEDURE IN** or **INOUT** output parameter passes the **ref cursor** parameter (not supported).

```
gaussdb=# CREATE OR REPLACE PROCEDURE proc_sys_ref(IN c1 REFCURSOR)
IS
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
  temp NUMBER(4);
BEGIN
  IF c1%ISOPEN THEN
    RAISE NOTICE '%','OK';
  END IF;

  LOOP
    FETCH c1 INTO temp;
    RAISE NOTICE '%',c1%ROWCOUNT;
    EXIT WHEN c1%NOTFOUND;
  END LOOP;
END;
/
CREATE PROCEDURE

gaussdb=# CREATE OR REPLACE PROCEDURE proc_sys_call() AS
DECLARE
  c1 SYS_REFCURSOR;
```



```
 temp NUMBER(4);
BEGIN
 OPEN c1 FOR SELECT section_id FROM sections ORDER BY section_id;
 proc_sys_ref(c1);
END;
/
CREATE PROCEDURE

gaussdb=# CALL proc_sys_call();
ERROR:  Unsupported: ref_cursor parameter is not supported for autonomous transactions.
CONTEXT:  SQL statement "CALL proc_sys_ref(c1)"
PL/pgSQL function proc_sys_call() line 7 at PERFORM
gaussdb=# DROP PROCEDURE proc_sys_ref;
DROP PROCEDURE
```

c. The **FUNCTION RETURN** output parameter passes the **ref cursor** parameter (not supported).

```
gaussdb=# DROP PROCEDURE IF EXISTS proc_sys_ref;
gaussdb=# CREATE OR REPLACE FUNCTION proc_sys_ref() RETURN SYS_REFCURSOR IS
DECLARE
 PRAGMA AUTONOMOUS_TRANSACTION;
       c1 SYS_REFCURSOR;
BEGIN
 OPEN c1 FOR SELECT section_id FROM sections ORDER BY section_id;
 return c1;
END;
/
ERROR:  Autonomous function do not support ref cursor as return types or out, inout arguments.
DETAIL:  N/A
```

d. The **FUNCTION OUT** output parameter passes the **ref cursor** parameter (not supported).

```
gaussdb=# DROP FUNCTION IF EXISTS proc_sys_ref;
gaussdb=# CREATE OR REPLACE FUNCTION proc_sys_ref(C1 out SYS_REFCURSOR)
gaussdb-# return SYS_REFCURSOR
gaussdb-# IS
gaussdb$# declare
gaussdb$#   PRAGMA AUTONOMOUS_TRANSACTION;
gaussdb$# BEGIN
gaussdb$#  OPEN C1 FOR SELECT section_ID FROM sections ORDER BY section_ID;
gaussdb$#  return 1;
gaussdb$# END;
gaussdb$# /
ERROR:  Autonomous function do not support ref cursor as return types or out, inout arguments.
DETAIL:  N/A
```

e. On the client (such as gsql and JDBC), the autonomous transaction PROCEDURE with the output parameter **ref cursor** is directly called. (Not supported. In this case, cursor data cannot be read.)

```
gaussdb=# CREATE OR REPLACE PROCEDURE proc_sys_ref(OUT c1 REFCURSOR)
IS
DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
OPEN c1 FOR SELECT section_id FROM sections ORDER BY section_id;
END;
/
CREATE PROCEDURE
gaussdb=# begin;
BEGIN
gaussdb=# call proc_sys_ref(null);
        c1
--------------------
 <unnamed portal 1>
(1 row)

gaussdb=# fetch "<unnamed portal 1>";
ERROR:  cursor "<unnamed portal 1>" does not exist
gaussdb=# end;
```



```
ROLLBACK
gaussdb=# DROP PROCEDURE proc_sys_ref;
DROP PROCEDURE
-- Drop the table.
gaussdb=# DROP TABLE sections;
DROP TABLE
```

- The autonomous transaction function cannot directly return the record type or the **out** output parameter and the record type at the same time.

```
gaussdb=# CREATE TABLE test_in (id INT,a DATE);
CREATE TABLE

gaussdb=# CREATE OR REPLACE FUNCTION autonomous_out()
RETURNS RECORD
LANGUAGE PLPGSQL AS $$
DECLARE PRAGMA AUTONOMOUS_TRANSACTION;
        r1 RECORD;
BEGIN
   DBE_OUTPUT.PRINT_LINE('THIS IS IN AUTONOMOUS_F_139_7');
   TRUNCATE test_in;
   INSERT INTO test_in VALUES (1,'1909-01-01');
   SELECT * INTO r1 FROM test_in;
RETURN r1;
END;
$$;
CREATE FUNCTION

gaussdb=# SELECT ok.id,ok.a FROM autonomous_out() AS ok(id INT,a DATE);
ERROR:  unrecognized return type for PLSQL function.

gaussdb=# CREATE TYPE rec IS (e1 INTEGER, e2 VARCHAR2);
CREATE TYPE
gaussdb=# CREATE OR REPLACE FUNCTION func(ele3 INOUT VARCHAR2) RETURN rec AS
i INTEGER;
ele1 rec;
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
NULL;
RETURN ele1;
END;
/
CREATE FUNCTION

gaussdb=# CALL func(1);
 e1 | e2
----+----
    |
(1 row)

gaussdb=# DROP TABLE test_in;
DROP TABLE
gaussdb=# DROP FUNCTION autonomous_out;
DROP FUNCTION
gaussdb=# DROP FUNCTION func;
DROP FUNCTION
```

- The isolation level of an autonomous transaction cannot be changed.

```
gaussdb=# CREATE OR REPLACE PROCEDURE auto_func(r INT)
AS
DECLARE
  a INT;
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
 a:=r;
END;
/
CREATE PROCEDURE
gaussdb=# call auto_func(1);
ERROR:  ERROR:  SET TRANSACTION ISOLATION LEVEL must be called before any query
```



```
CONTEXT:  SQL statement "SET TRANSACTION ISOLATION LEVEL SERIALIZABLE"
PL/pgSQL function auto_func(integer) line 6 at SQL statement
referenced column: auto_func
gaussdb=# DROP FUNCTION auto_func;
DROP FUNCTION
```

- Autonomous transactions do not support the **setof** return type.

```
gaussdb=# CREATE TABLE test_in (id INT,a DATE);
CREATE TABLE
gaussdb=# CREATE TABLE test_main (id INT,a DATE);
CREATE TABLE
gaussdb=# INSERT INTO test_main VALUES (1111,'2021-01-01'),(2222,'2021-02-02');
INSERT 0 2
gaussdb=# TRUNCATE test_in,test_main;
TRUNCATE TABLE
gaussdb=# CREATE OR REPLACE FUNCTION autonomous_f_022(num1  INT) RETURNS SETOF test_in
LANGUAGE PLPGSQL AS $$
DECLARE
count INT :=3;
test_row test_in%ROWTYPE;
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
   WHILE TRUE
   LOOP
   IF count=3 THEN
   NULL;
   ELSE
   IF count=2 THEN
      INSERT INTO test_main VALUES (count,'2021-03-03');
      GOTO pos1;
   END IF;
   END IF;
   count=count-1;
   END LOOP;
   INSERT INTO test_main VALUES (1000,'2021-04-04');
    <<pos1>>
    FOR test_row IN SELECT * FROM test_main
    LOOP
       RETURN NEXT test_row;
      END LOOP;
     RETURN;
END;
$$;
ERROR:  Autonomous transactions do not support RETURN SETOF.
DETAIL:  N/A

gaussdb=# DROP TABLE test_main;
DROP TABLE
gaussdb=# DROP TABLE test_in;
DROP TABLE
```

11.2 Stored Procedure Supporting Autonomous Transaction

An autonomous transaction can be defined in a stored procedure. The identifier of an autonomous transaction is **PRAGMA AUTONOMOUS_TRANSACTION**. The syntax of an autonomous transaction is the same as that of creating a stored procedure. For details, see **CREATE PROCEDURE**. The following is an example.

```
-- Create a table.
gaussdb=# CREATE TABLE t2(a INT, b INT);
CREATE TABLE
gaussdb=# INSERT INTO t2 VALUES(1,2);
INSERT 0 1
gaussdb=# SELECT * FROM t2;
 a | b
```



```
---+---
 1 | 2
(1 row)

-- Create a stored procedure that contains an autonomous transaction.
gaussdb=# CREATE OR REPLACE PROCEDURE autonomous_4(a INT, b INT)  AS
DECLARE
    num3 INT := a;
    num4 INT := b;
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    INSERT INTO t2 VALUES(num3, num4);
    DBE_OUTPUT.PRINT_LINE('JUST USE CALL.');
END;
/
CREATE PROCEDURE
-- Create a common stored procedure that calls an autonomous transaction stored procedure.
gaussdb=# CREATE OR REPLACE PROCEDURE autonomous_5(a INT, b INT)  AS
DECLARE
BEGIN
    DBE_OUTPUT.PRINT_LINE('JUST NO USE CALL.');
    INSERT INTO t2 VALUES(666, 666);
    autonomous_4(a,b);
    ROLLBACK;
END;
/
CREATE PROCEDURE
-- Call a common stored procedure.
gaussdb=# SELECT autonomous_5(11,22);
JUST NO USE CALL.
JUST USE CALL.
 autonomous_5
--------------

(1 row)

-- View the table result.
gaussdb=# SELECT * FROM t2 ORDER BY a;
 a  | b
----+----
  1 |  2
 11 | 22
(2 rows)

gaussdb=# DROP TABLE t2;
DROP TABLE
gaussdb=# DROP PROCEDURE autonomous_4;
DROP PROCEDURE
gaussdb=# DROP PROCEDURE autonomous_5;
DROP PROCEDURE
```

In the preceding example, a stored procedure containing an autonomous transaction is finally executed in a transaction block to be rolled back, which directly illustrates a characteristic of the autonomous transaction, that is, rollback of the primary transaction does not affect content that has been committed by the autonomous transaction.

11.3 Anonymous Block Supporting Autonomous Transaction

An autonomous transaction can be defined in an anonymous block. The identifier of an autonomous transaction is **PRAGMA AUTONOMOUS_TRANSACTION**. The syntax of an autonomous transaction is the same as that of creating an

anonymous block. For details, see **Anonymous Blocks**. The following is an example.

```
gaussdb=# CREATE TABLE t1(a INT ,B TEXT);
CREATE TABLE

gaussdb=# START TRANSACTION;
START TRANSACTION

gaussdb=# DECLARE
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    DBE_OUTPUT.PRINT_LINE('JUST USE CALL.');
    INSERT INTO t1 VALUES(1,'YOU ARE SO CUTE,WILL COMMIT!');
END;
/
JUST USE CALL.
ANONYMOUS BLOCK EXECUTE

gaussdb=# INSERT INTO t1 VALUES(1,'YOU WILL ROLLBACK!');
INSERT 0 1
gaussdb=# ROLLBACK;
ROLLBACK

gaussdb=# SELECT * FROM t1;
 a |             b
---+-----------------------------
 1 | YOU ARE SO CUTE,WILL COMMIT!
(1 row)

gaussdb=# DROP TABLE t1;
DROP TABLE
```

In the preceding example, an anonymous block containing an autonomous transaction is finally executed before a transaction block to be rolled back, which directly illustrates a characteristic of the autonomous transaction, that is, rollback of the primary transaction does not affect content that has been committed by the autonomous transaction.

11.4 Function Supporting Autonomous Transaction

An autonomous transaction can be defined in a function. The identifier of an autonomous transaction is **PRAGMA AUTONOMOUS_TRANSACTION**. The syntax of an autonomous transaction is the same as that of creating a function. For details, see **CREATE FUNCTION**. The following is an example.

```
gaussdb=# CREATE TABLE t4(a INT, b INT, c TEXT);
CREATE TABLE

gaussdb=# CREATE OR REPLACE FUNCTION autonomous_32(a INT ,b INT ,c TEXT) RETURN INT AS
DECLARE
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    INSERT INTO t4 VALUES(a, b, c);
    RETURN 1;
END;
/
CREATE FUNCTION

gaussdb=# CREATE OR REPLACE FUNCTION autonomous_33(num1 INT) RETURN INT AS
DECLARE
    num3 INT := 220;
    tmp INT;
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
```



```
    num3 := num3/num1;
    RETURN num3;
EXCEPTION
    WHEN DIVISION_BY_ZERO THEN
        SELECT autonomous_32(num3, num1, SQLERRM) INTO tmp;
        ROLLBACK;
        RETURN 0;
END;
/
CREATE FUNCTION

gaussdb=# SELECT autonomous_33(0);
 autonomous_33
---------------
             0
(1 row)

gaussdb=# SELECT * FROM t4;
  a  | b |        c
-----+---+------------------
 220 | 0 | division by zero
(1 row)

gaussdb=# DROP TABLE t4;
DROP TABLE
gaussdb=# DROP FUNCTION autonomous_32;
DROP FUNCTION
gaussdb=# DROP FUNCTION autonomous_33;
DROP FUNCTION
```

In the preceding example, a function containing an autonomous transaction is finally executed in a transaction block to be rolled back, which directly illustrates a characteristic of the autonomous transaction, that is, rollback of the primary transaction does not affect content that has been committed by the autonomous transaction.

11.5 Package Supporting Autonomous Transaction

An autonomous transaction can be defined in a stored procedure or function in a package. The identifier of an autonomous transaction is **PRAGMA AUTONOMOUS_TRANSACTION**. The syntax of an autonomous transaction is the same as that of creating a stored procedure or function in a package. For details, see **CREATE PACKAGE**. The following is an example.

```
-- Create a table.
gaussdb=# CREATE TABLE t2(a INT, b INT);
CREATE TABLE
gaussdb=# INSERT INTO t2 VALUES(1,2);
INSERT 0 1
gaussdb=# SELECT * FROM t2;
 a | b
---+---
 1 | 2
(1 row)

-- Create a stored procedure or function in a package that contains autonomous transactions.
gaussdb=# CREATE OR REPLACE PACKAGE autonomous_pkg AS
  PROCEDURE autonomous_4(a INT, b INT);
  FUNCTION autonomous_32(a INT ,b INT) RETURN INT;
END autonomous_pkg;
/
CREATE PACKAGE
gaussdb=# CREATE OR REPLACE PACKAGE BODY autonomous_pkg AS
PROCEDURE autonomous_4(a INT, b INT)  AS
DECLARE
```



```
 num3 INT := a;
 num4 INT := b;
 PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    INSERT INTO t2 VALUES(num3, num4);
END;
FUNCTION autonomous_32(a INT ,b INT) RETURN INT AS
DECLARE
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    INSERT INTO t2 VALUES(a, b);
    RETURN 1;
END;
END autonomous_pkg;
/
CREATE PACKAGE BODY

-- Create a common stored procedure that calls a stored procedure or function from a package that
contains autonomous transactions.
gaussdb=# CREATE OR REPLACE PROCEDURE autonomous_5(a INT, b INT)  AS
DECLARE
va INT;
BEGIN
    INSERT INTO t2 VALUES(666, 666);
    autonomous_pkg.autonomous_4(a,b);
        va := autonomous_pkg.autonomous_32(a + 1, b + 1);
    ROLLBACK;
END;
/
CREATE PROCEDURE
-- Call a common stored procedure.
gaussdb=# SELECT autonomous_5(11,22);
 autonomous_5
--------------

(1 row)

-- View the table result.
gaussdb=# SELECT * FROM t2 ORDER BY a;
 a  | b
----+----
  1 |  2
 11 | 22
 12 | 23
(3 rows)

gaussdb=# DROP TABLE t2;
DROP TABLE
```

In the preceding example, a stored procedure or function in a package containing autonomous transactions is finally executed in a transaction block that is rolled back, which directly illustrates a characteristic of the autonomous transaction, that is, rollback of the primary transaction does not affect content that has been committed by an autonomous transaction.

12 System Catalogs and System Views

12.1 Overview of System Catalogs and System Views

System catalogs store the structured metadata of GaussDB. They are the source of information used by GaussDB to control system running and are a core component of the database system.

System views provide ways to query system catalogs and internal database status.

System catalogs and system views are visible to either system administrators or all users. Some of the following system catalogs and views have marked the need of administrator permissions. They are accessible only to administrators.

You can delete and rebuild the catalogs, add columns to them, and insert and update values in them, but doing so may make system information inconsistent and cause system faults. Generally, users should not modify system catalogs or system views, or rename their schemas. They are automatically maintained by the system.

NOTICE

- You are advised not to modify the permissions on system catalogs or system views.
- Do not add, delete, or modify system catalogs because doing so will result in exceptions or even database unavailability.
- For details about column types in system catalogs and system views, see **Data Types**.
- For ADM views, the accessed objects are all objects of this type in the database. Therefore, unified permission management is required for ADM views. By default, only the system administrator has the permission to access ADM views. Some ADM view data comes from public and non-sensitive columns in base tables. For DB views, the objects that the current user has the permission to access in the database can be queried. Common users can access these views. For MY views, objects to which the current user belongs can be queried. Common users can access these views.

12.2 System Catalogs

12.2.1 Partitioned Table

12.2.1.1 PG_PARTITION

PG_PARTITION records all partitioned tables, table partitions, and index partitions in the database. Partitioned index information is not stored in the system catalog PG_PARTITION. The partitioned table does not have actual physical files. Therefore, pg_partition does not record information such as **relfilenode**, **relpages**, **reltuples**, **reltoastrelid**, and **reltoastidxid**.

Table 12-1 PG_PARTITION columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
relname	name	Names of the partitioned tables, table partitions, TOAST tables on table partitions, and index partitions.
parttype	"char"	Object type. ● **'r'**: partitioned table. ● **'p'**: table partition. ● **'s'**: table subpartition. ● **'x'**: index partition.
parentid	oid	● OID of the partitioned table in PG_CLASS when the object is a partitioned table or a partition. ● OID of the level-1 partition in PG_PARTITION when the object is a level-2 partition. ● OID of the partitioned index when the object is an index partition.
rangenum	integer	Reserved column.
intervalnum	integer	Reserved column.

Name	Type	Description
partstrategy	"char"	Partitioning policy of the partitioned table. ● **'r'**: range partition. ● **'i'**: interval partition. ● **'l'**: list partition. ● **'a'**: automatically extended list partition. ● **'h'**: hash partition. ● **'n'**: no partition policy. The object is not a table partition.
relfilenode	oid	Physical storage locations of the table partition, index partition, and TOAST table on the table partition.
reltablespace	oid	OID of the tablespace containing the table partition, index partition, and TOAST table on the table partition.
relpages	double precision	Statistics: numbers of data pages of the table partition and index partition.
reltuples	double precision	Statistics: numbers of tuples of the table partition and index partition.
relallvisible	integer	Statistics: number of visible data pages of the table partition and index partition.
reltoastrelid	oid	OID of the TOAST table corresponding to the table partition.
reltoastidxid	oid	OID of the TOAST table index corresponding to the table partition.
indextblid	oid	OID of the table partition corresponding to the index partition.
indisusable	Boolean	Specifies whether the index partition is available.
relfrozenxid	xid32	Frozen transaction ID. To ensure forward compatibility, this column is reserved. The **relfrozenxid64** column is added to record the information.
intspnum	integer	Number of tablespaces that the interval partition belongs to.
partkey	int2vector	Column number of the partition key.
intervaltablespace	oidvector	Tablespace that the interval partition belongs to. Interval partitions fall in the tablespaces in the round-robin manner.

Name	Type	Description
interval	text[]	Interval value of the interval partition.
boundaries	text[]	Upper boundary of the range partition and interval partition.
transit	text[]	Transit of the interval partition.
reloptions	text[]	Storage attribute of a partition used for collecting online scale-out information. Same as **pg_class.reloptions**, it is expressed in a string in the format of *keyword*=*value*.
relfrozenxid64	xid	Frozen transaction ID.
relminmxid	xid	Frozen multi-transaction ID.
partitionno	integer	Used for maintaining the partition map of a partitioned table. ● If the object is a partition, this field indicates the partition ID, which starts from 1 in ascending order. ● If the object is a partitioned table, this field indicates the maximum partition ID and a negative value is used for special meaning. The value increases with the DDL syntax of some partitions. ● If the object is of other types, this field is null and has no meaning. **partitionno** is a permanent auto-increment column, which can be reset or reclaimed by using the syntax ALTER TABLE t_name RESET PARTITION.
subpartitionno	integer	Used for maintaining the map of level-2 partitions in a partitioned table. ● If the object is a level-2 partition, this field indicates the level-2 partition ID, which starts from 1 in ascending order. ● If the object is a level-1 partition in a level-2 partitioned table, this field indicates the maximum level-2 partition ID and a negative value is used for special meaning. This value increases with the DDL syntax of some partitions. ● If the object is of other types, this field is null and has no meaning. **subpartitionno** is a permanent auto-increment column, which can be reset or reclaimed by using the syntax ALTER TABLE t_name RESET PARTITION.

12.2.2 OLTP Table Compression

12.2.2.1 GS_ILM

The GS_ILM system catalog provides the main information about the ILM policy, including the policy name, policy owner, policy type, policy ID, and policy status.

Table 12-2 GS_ILM columns

Name	Type	Description
pidx	integer	Policy sequence number, which is globally unique and starts with 1.
creator	oid	Policy owner.
name	name	Policy name. Currently, the policy name cannot be customized. The default name is p+pidx.
ptype	"char"	Policy type: • **m**: data movement.
flag	smallint	Value range: • **0**: The policy is enabled. • **1**: The policy is disabled.

12.2.2.2 GS_ILM_JOBDETAIL

The GS_ILM_JOBDETAIL system catalog records policy action types and judgment conditions.

Table 12-3 GS_ILM_JOBDETAIL columns

Name	Type	Description
taskoid	bigint	OID of an ADO task.
jobtype	"char"	Current job type: compression ('c').
jobstatus	smallint	Current job status: • **1**: JOB CREATED • **2**: COMPLETED SUCCESSFULLY • **3**: FAILED • **4**: STOPPED • **5**: JOB CREATION FAILED

Name	Type	Description
jobname	text	The ADO job name is unique in the table and is joined with **job_name** in the PG_JOB system catalog.
starttime	timestamp with time zone	Start time.
completetime	timestamp with time zone	Completion time.
payload	text	PL/SQL code of the current job.
statistics	text	Statistics of the current job.
comments	text	Description of the current job execution, such as the failure cause.

12.2.2.3 GS_ILM_OBJECT

GS_ILM_OBJECT records the relationships between data objects and policies as well as the scheduling information of policies on the data objects. When a policy is set for a partitioned table, this catalog generates an independent record for each partition or subpartition.

Table 12-4 GS_ILM_OBJECT columns

Name	Type	Description
pidx	integer	Policy ID.
objoid	oid	OID of a logical data object which has a partitioned table and partitions with level-2 partitions.
dataobjoid	oid	OID of a physical data object, which is the entity that needs to execute the policy and the leaf of the tree-level partition relationship.
objtype	"char"	Current data object type. The options are as follows: • Table ('r') • Partition ('p') • Level-2 partition ('s')
origobjoid	oid	OID of the original object that defines the policy.

Name	Type	Description
origobjtype	"char"	Type of the original data object that defines the policy. The options are as follows: • Table ('r') • Partition ('p') • Level-2 partition ('s')
lastchktime	timestamp with time zone	Time of the last ADO task.
lastexetime	timestamp with time zone	Time of the last ADO job.
roundcnt	smallint	Number of successful rounds.
failcnt	smallint	Number of failures.
lastjobstatus	"char"	Execution status of the last job.
lastroundstarttime	timestamp with time zone	Start time of the full table scan.
lastjobblkid	bigint	ID of the block that has been processed by the job last time.
flag	smallint	Value range: • **0**: enabled. • **1**: disabled.

12.2.2.4 GS_ILM_PARAM

GS_ILM_PARAM is a feature parameter table, which records the parameters related to the ILM feature, such as the control parameters of background scheduling.

Table 12-5 GS_ILM_PARAM columns

Name	Type	Description
idx	smallint	Parameter ID.
name	name	Parameter name.
value	double precision	Parameter value.

Table 12-6 GS_ILM_PARAM feature parameter range

Parameter ID	Parameter Value	Description
1	EXECUTION_INTERVAL	Specifies the frequency of executing an ADO task, in minutes. The default value is **15**. The value is an integer or floating-point number greater than or equal to 1 and less than or equal to 2147483647. The value is rounded down.
2	RETENTION_TIME	Specifies the retention period of ADO-related history records, in days. The default value is **30**. The value is an integer or floating-point number greater than or equal to 1 and less than or equal to 2147483647. The value is rounded down.
7	ENABLE	Specifies the background scheduling status which cannot be modified in this interface. Otherwise, the message "Invalid argument value, ENABLED should be change by calling DBE_ILM_ADMIN.ENABLE_ILM and DBE_ILM_ADMIN.DISABLE_ILM" is displayed. Instead, use disable_ilm() and enable_ilm() to modify it.
11	POLICY_TIME	Specifies whether the time unit of ADO is day or second. The time unit second is used only for testing. The value can be **ILM_POLICY_IN_SECONDS** or **ILM_POLICY_IN_DAYS** (default value).
12	ABS_JOBLIMIT	Specifies the maximum number of ADO jobs generated by an ADO task. The value is an integer or floating-point number greater than or equal to 0 and less than or equal to 2147483647. The value is rounded down.
13	JOB_SIZELIMIT	Specifies the maximum number of bytes that can be processed by a single ADO job. The unit is MB. The value is an integer or floating-point number greater than or equal to 0 and less than or equal to 2147483647. The value is rounded down.
14	WIND_DURATION	Specifies the maintenance window duration, in minutes. The default value is 240 minutes (4 hours). The value is an integer greater than or equal to 0 and less than 1440 (24 hours).

Parameter ID	Parameter Value	Description
15	BLOCK_LIMITS	Specifies the upper limit of the instance-level row-store compression rate. The default value is **40**. The value ranges from 0 to 10000, in block/ms, indicating the maximum number of blocks that can be compressed per millisecond. **0** indicates that the rate is not limited.

12.2.2.5 GS_ILM_POLICY

The GS_ILM_POLICY system catalog records policy action types and judgment conditions.

Table 12-7 GS_ILM_POLICY columns

Name	Type	Description
pidx	integer	Policy ID.
action	"char"	Policy action. Only compression ('c') is supported.
ctype	"char"	Compression type. Only advanced row compression ('a') is supported.
condition	"char"	Condition type. Only LAST MODIFICATION('m') is supported.
days	smallint	Number of judgment days.
scope	"char"	Policy scope. Only row ('r') is supported.
predicate	pg_node_tree	Policy row-level expression.

12.2.2.6 GS_ILM_TASK

The GS_ILM_TASK system catalog records ADO task information, such as the creation time, start time, end time, and execution status.

Table 12-8 GS_ILM_TASK columns

Name	Type	Description
taskoid	bigint	OID of an ADO task.
creator	oid	Creator of an ADO task.

Name	Type	Description
createtime	timestamp with time zone	Creation time.
starttime	timestamp with time zone	Start time.
completetime	timestamp with time zone	Completion time.
executestate	smallint	Value range: • **1**: 'INACTIVE' • **2**: 'ACTIVE' • **3**: 'COMPLETED'
flag	smallint	Reserved column.

12.2.2.7 GS_ILM_TASKDETAIL

The GS_ILM_TASKDETAIL system catalog records the evaluation result of a specified data object and policy.

Table 12-9 GS_ILM_TASKDETAIL columns

Name	Type	Description
pidx	integer	Policy ID.
objoid	oid	Data OID.
objtype	"char"	Data object type.
taskoid	bigint	OID of an ADO task.
evalresult	smallint	Evaluation result: • **0**: SELECTED FOR EXECUTION. The evaluation is passed. • **1**: PRECONDITION NOT SATISFIED. The condition is not met. • **2**: JOB ALREADY EXISTS. The job already exists.
jobname	text	ADO job name generated after the evaluation is passed is joined with the **jobname** column in the **GS_ILM_JOBDETAIL** system catalog.

12.2.2.8 GS_ILM_TICKER

The GS_ILM_TICKER system catalog records the mapping between LSNs and time. A maximum of 3650 lines of records can be retained.

Table 12-10 GS_ILM_TICKER columns

Name	Type	Description
ilm_ticker_lsn	bigint	LSN when the dotting operation is triggered.
ilm_ticker_time	timestamp with time zone	Time when the dotting operation is triggered.

12.2.3 Encrypted Equality Query

12.2.3.1 GS_CLIENT_GLOBAL_KEYS

GS_CLIENT_GLOBAL_KEYS records information about the CMK in the encrypted equality feature. Each record corresponds to a CMK.

Table 12-11 GS_CLIENT_GLOBAL_KEYS columns

Name	Type	Description
oid	oid	Row identifier (hidden column)
global_key_name	name	CMK name
key_namespace	oid	Namespace OID that contains the CMK
key_owner	oid	CMK owner
key_acl	aclitem[]	Access permissions that the key should have on creation
create_date	timestamp without time zone	Time when the key is created

12.2.3.2 GS_CLIENT_GLOBAL_KEYS_ARGS

GS_CLIENT_GLOBAL_KEYS_ARGS records the metadata about the CMK in the encrypted equality feature. Each record corresponds to a key-value pair of the CMK.

Table 12-12 GS_CLIENT_GLOBAL_KEYS_ARGS columns

Name	Type	Description
oid	oid	Row identifier (hidden column)
global_key_id	oid	CMK OID
function_name	name	The value is **encryption**.
key	name	CMK metadata name
value	bytea	Value of the CMK metadata name

12.2.3.3 GS_COLUMN_KEYS

GS_COLUMN_KEYS records information about the CEK in the encrypted equality feature. Each record corresponds to a CEK.

Table 12-13 GS_COLUMN_KEYS columns

Name	Type	Description
oid	oid	Row identifier (hidden column)
column_key_name	name	CEK name
column_key_distributed_id	oid	ID obtained based on the hash value of the fully qualified domain name (FQDN) of the CEK
global_key_id	oid	Foreign key, which is the CMK OID
key_namespace	oid	Namespace OID that contains the CEK
key_owner	oid	CEK owner
create_date	timestamp without time zone	Time when the CEK is created
key_acl	aclitem[]	Access permissions that the CEK should have on creation

12.2.3.4 GS_COLUMN_KEYS_ARGS

GS_COLUMN_KEYS_ARGS records the metadata about the CMK in the encrypted equality feature. Each record corresponds to a key-value pair of the CMK.

Table 12-14 GS_COLUMN_KEYS_ARGS columns

Name	Type	Description
oid	oid	Row identifier (hidden column)
column_key_id	oid	CEK OID
function_name	name	The value is **encryption**.
key	name	CEK metadata name
value	bytea	Value of the CEK metadata name

12.2.3.5 GS_ENCRYPTED_COLUMNS

GS_ENCRYPTED_COLUMNS records information about encrypted columns in the encrypted equality feature. Each record corresponds to an encrypted column.

Table 12-15 GS_ENCRYPTED_COLUMNS columns

Name	Type	Description
oid	oid	Row identifier (hidden column)
rel_id	oid	Table OID
column_name	name	Name of an encrypted column.
column_key_id	oid	Foreign key, which is the CEK OID
encryption_type	tinyint	Encryption type. The value can be **2 (DETERMINISTIC)** or **1 (RANDOMIZED)**.
data_type_original_oid	oid	ID of the original data type of the encrypted column. For details about the value, see **oid** in the **PG_TYPE** system catalog.
data_type_original_mod	integer	Original data type modifier of the encrypted column. For details about the value, see **atttypmod** in the **PG_ATTRIBUTE** system catalog. The value of **data_type_original_mod** is generally **–1** when data types are not specific.
create_date	timestamp without time zone	Time when an encrypted column is created

12.2.3.6 GS_ENCRYPTED_PROC

GS_ENCRYPTED_PROC provides information such as the parameters of encrypted functions and stored procedure functions, original data type of return values, and encrypted columns.

Table 12-16 GS_ENCRYPTED_PROC columns

Name	Type	Description
oid	oid	Row identifier (hidden column)
func_id	oid	OID of the function, corresponding to the OID row identifier in the **PG_PROC** system catalog.
prorettype_orig	integer	Original data type of the return value
last_change	timestamp without time zone	Last modification time of the encrypted function information
proargcachedcol	oidvector	OID of the encrypted column corresponding to the **INPUT** parameter of the function, corresponding to the OID row identifier in the **GS_ENCRYPTED_COLUMNS** system catalog.
proallargtypes_orig	oid[]	Original data type of all function parameters

12.2.4 Communications

12.2.4.1 PGXC_NODE

The **PGXC_NODE** system catalog stores information about database instance nodes. In the centralized system, only the definition of this table can be queried.

Table 12-17 PGXC_NODE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
node_name	name	Node name.

Name	Type	Description
node_type	"char"	Node type. • **C**: CN • **D**: DN • **S**: standby node
node_port	integer	Port number of the node.
node_host	name	Host name or IP address of a node. (If a virtual IP address is configured, its value is a virtual IP address.)
node_port1	integer	Port number of a replication node.
node_host1	name	Host name or IP address of a replication node. (If a virtual IP address is configured, its value is a virtual IP address.)
hostis_primary	Boolean	Specifies whether a primary/standby switchover occurs on the current node. • **t** (true): yes • **f** (false): no
nodeis_primary	Boolean	Specifies whether the current node is preferred to execute non-query operations in the **replication** table. • **t** (true): yes • **f** (false): no
nodeis_preferred	Boolean	Specifies whether the current node is preferred to execute queries in the **replication** table. • **t** (true): yes • **f** (false): no
node_id	integer	Node identifier. The value is obtain by calculating the value of **node_name** using the hash function.
sctp_port	integer	Port used by the TCP proxy communications library of the primary node to listen to the data channel. (The SCTP communications library is no longer supported in the current version.)
control_port	integer	Port used by the TCP proxy communications library of the primary node to listen to the control channel.
sctp_port1	integer	Port used by the TCP proxy communications library of the standby node to listen to the data channel. (The SCTP communications library is no longer supported in the current version.)

Name	Type	Description
control_port1	integer	Port used by the TCP proxy communications library of the standby node to listen to the control channel.
nodeis_central	Boolean	Specifies whether the current node is a CN. It is invalid for DNs. • **t** (true): yes • **f** (false): no
nodeis_active	Boolean	Specifies whether the current node is normal. It is invalid for DNs. • **t** (true): yes • **f** (false): no

12.2.5 Ledger Database

12.2.5.1 GS_GLOBAL_CHAIN

GS_GLOBAL_CHAIN records information about modification operations performed by users on the tamper-proof user table. Each record corresponds to a table-level modification operation. Users with the audit administrator permission can query this system catalog, but no user is allowed to modify this system catalog.

Table 12-18 GS_GLOBAL_CHAIN columns

Name	Type	Description
blocknum	bigint	Block number, which is the sequence number of the current user operation recorded in the ledger
dbname	name	Name of the database, to which the modified tamper-proof user table belongs
username	name	Username, which is the name of the user who performs the operation of modifying the user table
starttime	timestamp with time zone	Time when a user performs an operation.
relid	oid	OID of the modified tamper-proof user table
relnsp	name	Schema name, which is the name of the schema to which the modified tamper-proof user table belongs.
relname	name	User table name, which is the name of the modified tamper-proof user table

Name	Type	Description
relhash	hash16	Table-level hash change amount generated by the current operation
globalhash	hash32	Global digest, which is calculated based on the information of the current row and the **globalhash** of the previous row. It connects the entire table to verify the integrity of GS_GLOBAL_CHAIN data.
txcommand	text	SQL statement whose operations are recorded

12.2.6 SPM

12.2.6.1 GS_SPM_SQL

GS_SPM_SQL is a system catalog used to store SPM information. Users with sysadmin permissions can read the system catalog, but only initial users can write the system catalog.

Table 12-19 GS_SPM_SQL columns

Name	Type	Description
sql_namespace	oid	Schema OID.
sql_hash	bigint	Unique ID of the SQL statement in the SPM.
sql_text	text	SQL text string.
param_num	integer	Number of parameters required for SQL execution.
user	oid	OID of the login user.
creation_time	timestamp with time zone	Time when a record is created.

12.2.6.2 GS_SPM_BASELINE

GS_PLAN_BASELINE is a system catalog used to store baseline information. Users with sysadmin permissions can read the system catalog, but only initial users can write the system catalog.

Table 12-20 GS_SPM_BASELINE columns

Name	Type	Description
sql_namespace	oid	Schema OID.
sql_hash	bigint	Unique ID of the SQL statement in the SPM.
plan_hash	bigint	Unique ID of a plan in the current SQL statement.
outline	text	Outline text, which can be used to fix a group of hints of the current plan.
cost	double precision	Total plan cost.
user	oid	User who creates the baseline.
status	integer	Baseline status. The options are as follows: • **0** (UNACC): indicates that the plan is not accepted. • **1** (ACC): indicates that the plan has been accepted. • **2** (FIXED): indicates a special ACC plan. The matching priority of this plan is higher than that of other ACC plan.
source	text	Baseline source.
gplan	Boolean	Determines whether the plan is a gplan.
creation_time	timestamp with time zone	Time when the baseline is created.
last_used_time	timestamp with time zone	Last time when a plan was used.
modification_time	timestamp with time zone	Time when the baseline is modified.
jump_intercept_cnt	bigint	Number of baseline interception plan jumps.
invalid	Boolean	Specifies whether the current baseline is invalid.

12.2.6.3 GS_SPM_ID_HASH_JOIN

The GS_SPM_ID_HASH_JOIN system catalog stores the relationship between unique_sql_id and sql_hash. Users with sysadmin permissions can read the system catalog, but only initial users can write data to the system catalog.

Table 12-21 GS_SPM_ID_HASH_JOIN columns

Name	Type	Description
unique_sql_id	bigint	Unique ID of an SQL statement in the database.
sql_hash	bigint	Unique ID of the SQL statement in the SPM.

12.2.6.4 GS_SPM_PARAM

GS_SPM_PARAM is a system catalog used to store SQL parameters. Each SQL statement stores only one group of parameters. Users with the sysadmin permission can read the system catalog, but only initial users can write the system catalog.

Table 12-22 GS_SPM_PARAM columns

Name	Type	Description
sql_namespace	oid	Schema OID
sql_hash	bigint	Unique ID of the SQL statement in the SPM.
position	integer	Position index of a parameter in the SQL statement, starting from 0.
datatype	integer	OID of the parameter type.
datatype_string	text	Character string of a parameter type.
value_string	text	Character string of a parameter value.
is_null	Boolean	Determines whether a parameter value is **NULL**.
hash_value	bigint	Hash value of a parameter value.
user	oid	User who creates the record.
creation_time	timestamp with time zone	Time when a record is created.

12.2.6.5 GS_SPM_EVOLUTION

GS_SPM_EVOLUTION is a system catalog used to store plan evolution result. Users with sysadmin permissions can read the system catalog, but only initial users can write the system catalog.

Table 12-23 GS_SPM_EVOLTOIN columns

Name	Type	Description
sql_namespace	oid	Schema OID.
sql_hash	bigint	Unique ID of the SQL statement in the SPM.
plan_hash	bigint	Plan ID.
better	Boolean	Determines whether it is a positive evolution. ● **t** indicates positive evolution. ● **f** indicates negative evolution.
refer_plan	bigint	Plan hash used as a reference for report generation.
status	integer	Determines whether exceptions occur during the evolution. ● The value **0** indicates that no exception occurs. ● The value **1** indicates that an exception occurs.
reason	text	Content of the evolution report.
user	oid	User who generates the evolution result.
creation_time	timestamp with time zone	Time when the evolution result is created.

12.2.7 AI

12.2.7.1 GS_MODEL_WAREHOUSE

GS_MODEL_WAREHOUSE stores AI engine training models, including the models and detailed description of the training process.

Table 12-24 GS_MODEL_WAREHOUSE columns

Name	Data Type	Description
oid	oid	Hidden column
modelname	name	Unique constraint
modelowner	oid	OID of a model owner

Name	Data Type	Description
createtime	timestamp without time zone	Time when a model is created
processedtuples	integer	Number of tuples involved in training
discardedtuples	integer	Number of unqualified tuples not involved in training
preprocesstime	real	Data preprocessing time
exectime	real	Training duration
iterations	integer	Iteration round
outputtype	oid	OID of the output data type
modeltype	text	AI operator type
query	text	Query statement executed to create a model
modeldata	bytea	Stored binary model information
weight	real[]	Currently, this column applies only to GD operator models.
hyperparametersnames	text[]	Involved hyperparameter name
hyperparametersvalues	text[]	Hyperparameter value
hyperparametersoids	oid[]	OID of the data type corresponding to a hyperparameter
coefnames	text[]	Model parameter
coefvalues	text[]	Value of a model parameter
coefoids	oid[]	OID of the data type corresponding to a model parameter
trainingscoresname	text[]	Method used to measure model performance
trainingscoresvalue	real[]	Value used to measure model performance

Name	Data Type	Description
modeldescribe	text[]	Model description

12.2.7.2 GS_OPT_MODEL

GS_OPT_MODEL is a data table used when the AI engine is enabled to predict the planned time. It records the configurations, training results, features, corresponding system functions, and training history of machine learning models.

Table 12-25 GS_OPT_MODEL columns

Name	Type	Description
template_name	name	Template name of the machine learning model, which determines the interfaces called for training and prediction. Currently, only rlstm is implemented.
model_name	name	Model name. Each model corresponds to a set of parameters, training logs, and model coefficients in the AI engine online learning process. The name must be unique.
datname	name	Name of the database served by the model. Each model is specific to a single database. This parameter determines data used for training.
ip	name	IP address of the host where the AI engine is deployed
port	integer	Listening port number of the AI engine
max_epoch	integer	Maximum number of iterations in an epoch
learning_rate	real	Learning rate of model training. The default value **1** is recommended.

Name	Type	Description
dim_red	real	Number of model feature dimensions whose retention is reduced
hidden_units	integer	Number of neurons in the model's hidden layer. If the model cannot be converged for a long time, increase the value of this parameter.
batch_size	integer	Size of a batch in each iteration. It is recommended that the size be greater than or equal to the total training data volume to accelerate model convergence.
feature_size	integer	Length of the model feature, which is used to trigger retraining. This parameter is automatically updated after model training and does not need to be specified.
available	Boolean	Specifies whether the model is converged. This parameter does not need to be specified.
Is_training	Boolean	Specifies whether the model is being trained. This parameter does not need to be specified.
label	"char"[]	Target task of the model. ● S: startup time ● T: total time ● R: rows ● M: peak memory Currently, {S, T} or {R} is recommended due to model performance restrictions.

Name	Type	Description
max	bigint[]	Maximum value of each task label of the model, which is used to trigger retraining. This parameter does not need to be specified.
acc	real[]	Accuracy of each model task. This parameter does not need to be specified.
description	text	Model comment

12.2.7.3 GS_ABO_MODEL_STATISTIC

GS_ABO_MODEL_STATISTIC records the metadata, model name, and operator information of the ABO-based model estimation based on the feedback cardinality.

Table 12-26 GS_ABO_MODEL_STATISTIC columns

Name	Data Type	Description
oid	oid	Hidden column.
modelname	name	Model name, which is unique.
createtime	timestamp	Time when a model is created or updated.
groupkey	oid	Hash of the operator corresponding to the model.
joinkey	oid	Hash of the operator connection shape corresponding to the model.
fixed_selectivity	real	Fixed selectivity for operators that do not contain numeric conditions.
avg_qerror	real	Average errors of qerror in the model.
max_qerror	real	Maximum errors of qerror in the model.

Name	Data Type	Description
feedback_num	integer	Model training data volume.
status	integer	Model status. The value **0** indicates that the model is invalid. The value **1** indicates that the model is valid. The value **2** indicates that the model is to be updated. The value **3** indicates that the model is in the blocklist.
train_count	integer	Total number of model training times.
train_failure_count	integer	Number of model training failures.
prediction_count	integer	Total number of prediction times of a model.
predfiction_failure_count	integer	Number of model prediction failures.
rtable_ids	oid[]	Sequence of entering model conditions in the base table.
attnums	int2[]	Model condition input condition involves the column sequence.

12.2.8 Auditing

12.2.8.1 GS_AUDITING_POLICY

GS_AUDITING_POLICY records the main information about the unified audit. Each record corresponds to a design policy. Only the system administrator or security policy administrator can access this system catalog.

Table 12-27 GS_AUDITING_POLICY columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)

Name	Type	Description
polname	name	Policy name, which must be unique
polcomments	name	Policy description field, which records policy-related description information and is represented by the COMMENTS keyword
modifydate	timestamp without time zone	Latest timestamp when a policy is created or modified
polenabled	boolean	Specifies whether to enable a policy. • **t** (true): enabled. • **f** (false): disabled

12.2.8.2 GS_AUDITING_POLICY_ACCESS

GS_AUDITING_POLICY_ACCESS records the unified audit information about DML database operations. Only the system administrator or security policy administrator can access this system catalog.

Table 12-28 GS_AUDITING_POLICY_ACCESS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
accesstype	name	DML database operation type. For example, SELECT, INSERT, and DELETE.
labelname	name	Resource label name. This column corresponds to the **polname** column in the GS_AUDITING_POLICY system catalog.
policyoid	oid	OID of the audit policy, corresponding to the OID in the GS_AUDITING_POLICY system catalog
modifydate	timestamp without time zone	Latest creation or modification timestamp

12.2.8.3 GS_AUDITING_POLICY_FILTERS

GS_AUDITING_POLICY_FILTERS records the filtering policies about the unified audit. Each record corresponds to a design policy. Only the system administrator or security policy administrator can access this system catalog.

Table 12-29 GS_AUDITING_POLICY_FILTERS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
filtertype	name	Filter type. Currently, the value is **logical_expr**.
labelname	name	Name. Currently, the value is **logical_expr**.
policyoid	oid	OID of the audit policy, corresponding to the OID in the GS_AUDITING_POLICY system catalog
modifydate	timestamp without time zone	Latest creation or modification timestamp
logicaloperator	text	Logical character string of a filter criterion

12.2.8.4 GS_AUDITING_POLICY_PRIVILEGES

GS_AUDITING_POLICY_PRIVILEGES records the DDL database operations about the unified audit. Each record corresponds to a design policy. Only the system administrator or security policy administrator can access this system catalog.

Table 12-30 GS_AUDITING_POLICY_PRIVI columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
privilegetype	name	DDL database operation type. For example, CREATE, ALTER, and DROP.
labelname	name	Resource label name. This column corresponds to the **polname** column in the GS_AUDITING_POLICY system catalog.
policyoid	oid	This column corresponds to the OID in the GS_AUDITING_POLICY system catalog.
modifydate	timestamp without time zone	Latest creation or modification timestamp

12.2.9 User and Permission Management

12.2.9.1 GS_DB_PRIVILEGE

GS_DB_PRIVILEGE records the granting of ANY permissions. Each record corresponds to a piece of authorization information.

Table 12-31 GS_DB_PRIVILEGE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
roleid	oid	User ID.
privilege_type	text	ANY permission of a user. For details about the value, see **Table 7-223**.
admin_option	boolean	Whether the ANY permission recorded in the **privilege_type** column can be re-granted. • **t**: yes. • **f**: no.

12.2.9.2 PG_DB_ROLE_SETTING

PG_DB_ROLE_SETTING records the default values of configuration items bound to each role and database when the database is running.

Table 12-32 PG_DB_ROLE_SETTING columns

Name	Type	Description
setdatabase	oid	Database corresponding to the configuration items (**0** if no database is specified)
setrole	oid	Role corresponding to the configuration items (**0** if no role is specified)
setconfig	text[]	Default value of runtime configuration items. Contact the administrator to configure it.

12.2.9.3 PG_DEFAULT_ACL

PG_DEFAULT_ACL records initial permissions assigned to newly created objects.

Table 12-33 PG_DEFAULT_ACL columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
defaclrole	oid	ID of the role associated with the permission
defaclnamespace	oid	Namespace associated with the permission (**0** if no ID)
defaclobjtype	"char"	Object type of the permission • **r** indicates a table or view. • **S** indicates a sequence. • **f** indicates a function. • **T** indicates a type. • **K** indicates the client master key. • **k** indicates the column encryption key.
defaclacl	aclitem[]	Access permissions that this type of object should have on creation

12.2.9.4 PG_RLSPOLICY

PG_RLSPOLICY records row-level security policies.

Table 12-34 PG_RLSPOLICY columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
polname	name	Name of an access control policy.
polrelid	oid	OID of the table object on which the row-level security policy takes effect.
polcmd	"char"	SQL operations affected by the row-level security policy.
polpermissive	Boolean	Attribute of the row-level security policy. • **t**: OR condition concatenation of expressions. • **f**: AND condition concatenation of expressions.
polroles	oid[]	OID list of users affected by the row-level security policy. If this parameter is not specified, all users are affected.

Name	Type	Description
polqual	pg_node_tree	Expression of the row-level security policy.

12.2.9.5 PG_SECLABEL

PG_SECLABEL records security labels on database objects.

See also **PG_SHSECLABEL**, which provides a similar function for security labels of database objects that are shared within one database.

Table 12-35 PG_SECLABEL columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object that this security label pertains to
classoid	oid	**PG_CLASS**.oid	OID of the system catalog where the object appears
objsubid	integer	-	Column number for a security label on a table column
provider	text	-	Label provider associated with the label
label	text	-	Security label applied to the object

12.2.9.6 PG_SHSECLABEL

PG_SHSECLABEL records security labels on shared database objects. Security labels can be manipulated with the **SECURITY LABEL** command.

For an easier way to view security labels, see **PG_SECLABELS**.

See also **PG_SECLABEL**, which provides a similar function for security labels involving objects within a single database.

Unlike most system catalogs, **PG_SHSECLABEL** is shared across all databases in the system. There is only one copy of **PG_SHSECLABEL** in the GaussDB system, not one per database.

Table 12-36 PG_SHSECLABEL columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object that this security label pertains to

Name	Type	Reference	Description
classoid	oid	PG_CLASS.oid	OID of the system catalog where the object appears
provider	text	-	Label provider associated with the label
label	text	-	Security label applied to the object

12.2.9.7 PG_USER_MAPPING

PG_USER_MAPPING records mappings from local users to remote.

This system catalog is accessible only to system administrators. Common users can query the **PG_USER_MAPPINGS** view.

Table 12-37 PG_USER_MAPPING columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
umuser	oid	PG_AUTHID.oid	OID of the local role being mapped (**0** if the user mapping is public)
umserver	oid	PG_FOREIGN_SERVER.oid	OID of the foreign server that contains the mapping
umoptions	text[]	-	User mapping specific options, expressed in a string in the format of keyword=value

12.2.9.8 PG_USER_STATUS

PG_USER_STATUS records the states of users who access the database. This system catalog is accessible only to system administrators.

Table 12-38 PG_USER_STATUS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
roloid	oid	ID of the role.
failcount	integer	Number of failed attempts.

Name	Type	Description
locktime	timestamp with time zone	By default, the creation date of the role is displayed. If the role is locked by the administrator or the role is locked because the number of login failures exceeds the threshold, the date when the role is locked is displayed.
rolstatus	smallint	Role state. • **0**: normal. • **1**: The role is locked for a specific period of time because the failed login attempts exceed the threshold. • **2**: The role is locked by the administrator.
permspace	bigint	Size of the permanent table storage space used by the role.
tempspace	bigint	Size of the temporary table storage space used by the role.
password expired	smallint	Specifies whether a password is valid. • **0**: The password is valid. • **1**: The password is invalid.

12.2.10 Connection and Authentication

12.2.10.1 PG_AUTHID

PG_AUTHID records information about database authentication identifiers (roles). The concept of users is contained in that of roles. A user is actually a role whose **rolcanlogin** has been set. Any role, whether its **rolcanlogin** is set or not, can use other roles as members.

For GaussDB, only one PG_AUTHID exists, which is not available for every database. This system catalog is accessible only to system administrators.

Table 12-39 PG_AUTHID columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
rolname	name	Role name.

Name	Type	Description
rolsuper	Boolean	Specifies whether a role is the initial system administrator with the highest permission. ● **t** (true): yes ● **f** (false): no
rolinherit	Boolean	Specifies whether a role automatically inherits permissions of roles of which it is a member. ● **t** (true): automatically inherited ● **f** (false): not automatically inherited
rolcreaterole	Boolean	Specifies whether a role can create more roles. ● **t** (true): yes ● **f** (false): no
rolcreatedb	Boolean	Specifies whether a role can create databases. ● **t** (true): yes ● **f** (false): no
rolcatupdate	Boolean	Specifies whether the role can directly update system catalogs. Only the initial system administrator whose **usesysid** is set to **10** has this permission. It is unavailable for other users. ● **t** (true): yes ● **f** (false): no
rolcanlogin	Boolean	Specifies whether the role can log in, that is, whether the role can be given as the initial session authorization identifier. ● **t** (true): yes ● **f** (false): no
rolreplication	Boolean	Specifies whether the role has the replication permission. ● **t** (true): yes ● **f** (false): no
rolauditadmin	Boolean	Specifies whether the role has the audit administrator permission. ● **t** (true): yes ● **f** (false): no

Name	Type	Description
rolsystemadmin	Boolean	Specifies whether the role has the system administrator permission. ● **t** (true): yes ● **f** (false): no
rolconnlimit	integer	Maximum number of concurrent connections that this role can make (valid for roles that can log in). The value **–1** indicates there is no limit.
rolpassword	text	Password ciphertext. If there is no password, the value is **NULL**.
rolvalidbegin	timestamp with time zone	Account validity start time (**NULL** if no start time).
rolvaliduntil	timestamp with time zone	Password expiry time (**NULL** if no expiration).
rolrespool	name	Resource pool that a user can use.
roluseft	Boolean	Specifies whether the role can perform operations on foreign tables. ● **t** (true): yes ● **f** (false): no
rolparentid	oid	OID of a group user to which the user belongs.
roltabspace	text	Maximum size of a user data table.
rolkind	"char"	Type of a user. ● **n**: common user ● **p**: permanent user
rolnodegroup	oid	Unsupported currently.
roltempspace	text	Maximum size of a user's temporary table, in KB.
rolspillspace	text	Maximum size of data that can be written to disks when a user executes a job, in KB.
rolexcpdata	text	Query rules that can be set by users (reserved).
rolmonitoradmin	Boolean	Specifies whether the role has the monitor administrator permission. ● **t** (true): yes ● **f** (false): no

Name	Type	Description
roloperatoradmin	Boolean	Specifies whether the role has the O&M administrator permission. • **t** (true): yes • **f** (false): no
rolpolicyadmin	Boolean	Specifies whether the role has the security policy administrator permission. • **t** (true): yes • **f** (false): no

12.2.10.2 PG_AUTH_HISTORY

PG_AUTH_HISTORY records the authentication history of a role. This system catalog is accessible only to system administrators.

Table 12-40 PG_AUTH_HISTORY columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
roloid	oid	ID of a role
passwordtime	timestamp with time zone	Time of password creation and change
rolpassword	text	Ciphertext of the role password. The encryption mode is determined by the GUC parameter **password_encryption_type**.

12.2.10.3 PG_AUTH_MEMBERS

PG_AUTH_MEMBERS records the membership between roles.

Table 12-41 PG_AUTH_MEMBERS columns

Name	Type	Description
roleid	oid	ID of a role that has a member
member	oid	ID of a role that is a member of ROLEID
grantor	oid	ID of a role that grants this membership
admin_option	Boolean	Whether a member can grant membership in ROLEID to others. The value can be **true** (yes) and **false** (no).

12.2.11 Dynamic Data Masking

12.2.11.1 GS_MASKING_POLICY

GS_MASKING_POLICY records the main information about dynamic data masking policies. Each record corresponds to a masking policy. Only the system administrator or security policy administrator can access this system catalog.

Table 12-42 GS_MASKING_POLICY columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
polname	name	Policy name, which must be unique
polcomments	name	Policy description field, which records policy-related description information and is represented by the COMMENTS keyword
modifydate	timestamp without time zone	Latest timestamp when a policy is created or modified
polenabled	boolean	Specifies whether to enable a policy. • **t** (true): enabled. • **f** (false): disabled

12.2.11.2 GS_MASKING_POLICY_ACTIONS

GS_MASKING_POLICY_ACTIONS records the masking actions of a masking policy in the dynamic data masking policies. One masking policy corresponds to one or more rows of records in the catalog. Only the system administrator or security policy administrator can access this system catalog.

Table 12-43 GS_MASKING_POLICY_ACTIONS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
actiontype	name	Name of a masking function used by a masking policy

Name	Type	Description
actparams	name	Parameter information transferred to a masking function
actlabelname	name	Name of a masked label
policyoid	oid	OID of a masking policy to which a record belongs, corresponding to the OID in GS_MASKING_POLICY
actmodifydate	timestamp without time zone	Latest timestamp when a record is created or modified

12.2.11.3 GS_MASKING_POLICY_FILTERS

GS_MASKING_POLICY_FILTERS records the user filter criteria corresponding to the dynamic data masking policies. The corresponding masking policy takes effect only when the user information meets the filter criteria. Only the system administrator or security policy administrator can access this system catalog.

Table 12-44 GS_MASKING_POLICY_FILTERS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
filtertype	name	Filter type. Currently, the value is **logical_expr**.
filterlabelname	name	Filter range. Currently, the value is **logical_expr**.
policyoid	oid	OID of the masking policy to which a user filter criterion belongs, which corresponds to the OID in GS_MASKING_POLICY
modifydate	timestamp without time zone	Latest timestamp when a user filter criterion is created or modified
logicaloperator	text	Polish notation of a filter criterion

12.2.12 DATABASE LINK

12.2.12.1 GS_DATABASE_LINK

GS_DATABASE_LINK stores database link information, mainly recording detailed information about database links. Only sysadmin can read this system catalog. Currently, the database link function is not supported.

Table 12-45 GS_DATABASE_LINK columns

Name	Type	Description
oid	oid	Unique ID of the current database link object (hidden attribute, which must be specified).
dlname	name	Name of the current database link.
dlowner	oid	ID of the owner of the current DATABASE LINK. If the owner is **public**, the value is **0**.
dlfdw	oid	OID of the foreign-data wrapper of the current DATABASE LINK.
dlcreater	oid	ID of the creator of the current database link.
options	text[]	Current DATABASE LINK connection information in the format of "keyword=value".
useroptions	text[]	User information used by the current DATABASE LINK to connect to the remote end, in the format of "keyword=value".
dlacl	aclitem[]	Current database link access permission.

12.2.13 Materialized Views

12.2.13.1 GS_MATVIEW

GS_MATVIEW provides information about each materialized view in the database.

Table 12-46 GS_MATVIEW columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
matviewid	oid	OID of a materialized view
mapid	oid	OID of a map table associated with a materialized view. Each map table corresponds to one materialized view. If a complete-refresh materialized view does not correspond to a map table, the value of this column is **0**.

Name	Type	Description
ivm	boolean	Type of a materialized view. The value **t** indicates a fast-refresh materialized view, and the value **f** indicates a complete-refresh materialized view.
needrefresh	boolean	Reserved column
refreshtime	timestamp without time zone	Last time when a materialized view was refreshed. If the materialized view is not refreshed, the value is null. This column is maintained only for fast-refresh materialized views. For complete-refresh materialized views, the value is null.

12.2.13.2 GS_MATVIEW_DEPENDENCY

GS_MATVIEW_DEPENDENCY provides association information about each fast-refresh materialized view, base table, and Mlog table in the database. The Mlog table corresponding to the base table does not exist in the complete-refresh materialized view. Therefore, no record is written into the Mlog table.

Table 12-47 GS_MATVIEW_DEPENDENCY columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
matviewid	oid	OID of a materialized view
relid	oid	OID of a base table of a materialized view
mlogid	oid	OID of a Mlog table which is the log table of a materialized view. Each Mlog table corresponds to one base table.
mxmin	integer	Reserved column

12.2.14 Other System Catalogs

12.2.14.1 GS_ASP

GS_ASP displays the persistent ACTIVE SESSION PROFILE samples. This catalog can be queried only in the system database but cannot be queried in the user database.

Table 12-48 GS_ASP columns

Name	Type	Description
sampleid	bigint	Sample ID.
sample_time	timestamp with time zone	Sampling time.
need_flush_sample	Boolean	Specifies whether the sample needs to be flushed to disks. • **t** (true): yes • **f** (false): no
databaseid	oid	Database ID.
thread_id	bigint	Thread ID.
sessionid	bigint	Session ID.
start_time	timestamp with time zone	Start time of a session.
event	text	Event name. For key events in the kernel, see **Table 12-412**, **15.3.67-Table 3 List of wait events corresponding to lightweight locks**, **Table 12-414**, and **Table 12-415**. For details about the impact of each transaction lock on services, see **LOCK**.
lwtid	integer	Lightweight thread ID of the current thread.
psessionid	bigint	Parent thread of the streaming thread.
tlevel	integer	Level of the streaming thread. The value corresponds to the level (ID) of the execution plan.
smpid	integer	Concurrent thread ID in SMP execution mode.
userid	oid	ID of a session user.
application_name	text	Name of an application.
client_addr	inet	IP address of a client.
client_hostname	text	Name of a client.
client_port	integer	TCP port number used by a client to communicate with the backend.
query_id	bigint	Debug query ID.

Name	Type	Description
unique_query_id	bigint	Unique query ID.
user_id	oid	User ID in the key of the unique query.
cn_id	integer	ID of the node that delivers the unique SQL statement. **cn_id** is in the key of the unique query.
unique_query	text	Standardized unique SQL text string.
locktag	text	Information of a lock that the session waits for, which can be parsed using **locktag_decode**.
lockmode	text	Mode of a lock that the session waits for. • **LW_EXCLUSIVE**: exclusive lock • **LW_SHARED**: shared lock • **LW_WAIT_UNTIL_FREE**: waits for the **LW_EXCLUSIVE** to be available
block_sessionid	bigint	Blocks a session from obtaining the session ID of a lock if the session is waiting for the lock.
wait_status	text	Provides more details about an event column.
global_sessionid	text	Global session ID.
xact_start_time	timestamp with time zone	Start time of the transaction.
query_start_time	timestamp with time zone	Time when the statement starts to be executed.

Name	Type	Description
state	text	Current transaction state. The options are as follows: • **active**: The backend is executing a query. • **idle in transaction**: The backend is in a transaction, but there is no statement being executed in the transaction. • **idle in transaction (aborted)**: The backend is in a transaction, but there are statements failed in the transaction. • **fastpath function call**: The backend is executing a fast-path function. • **disabled**: This state is reported if **track_activities** is disabled in this backend.
event_start_time	timestamp with time zone	Start time of a wait event.
current_xid	xid	ID of the current transaction.
top_xid	xid	ID of the top-level transaction that is being executed.

12.2.14.2 GS_DEPENDENCIES

The GS_DEPENDENCIES system catalog records object dependency information and has a one-to-many relationship with the **GS_DEPENDENCIES_OBJ** table.

Table 12-49 GS_DEPENDENCIES columns

Name	Type	Description
schemaname	name	Name of a namespace
packagename	name	Name of a package

Name	Type	Description
refobjpos	integer	Position of the referenced object • **1**: type • **2**: package header • **4**: function header • **8**: function body • **16**: package body • **32**: view
refobjoid	oid	OID of the referenced object
objectname	text	Name of the dependent object

12.2.14.3 GS_DEPENDENCIES_OBJ

The GS_DEPENDENCIES_OBJ system catalog records the detailed information about the referenced object.

Table 12-50 GS_DEPENDENCIES_OBJ columns

Name	Type	Description
schemaname	name	Name of a namespace
packagename	name	Name of a package
type	integer	Type of the referenced object • **1**: unknown type • **2**: variable • **3**: type • **4**: function • **5**: view • **6**: function header
name	text	Name of the referenced object
objnode	pg_node_tree	Detailed information about the referenced object

12.2.14.4 GS_GLOBAL_CONFIG

GS_GLOBAL_CONFIG records the parameter values specified by users during database instance initialization. In addition, it also stores weak passwords set by users. Initial database users can write, modify, and delete parameters in system catalogs by using ALTER and DROP. Only the initial user, system administrator, and security administrator can access this system catalog. Other users do not have such permission.

Table 12-51 GS_GLOBAL_CONFIG columns

Name	Type	Description
name	name	Specifies the preset parameter name, weak password name, or parameter required by users during database instance initialization.
value	text	Specifies the preset parameter value, weak password value, or parameter value required by users during database instance initialization.

12.2.14.5 GS_JOB_ARGUMENT

GS_JOB_ARGUMENT provides the parameter attributes of DBE_SCHEDULER scheduled tasks and programs.

Table 12-52 GS_JOB_ARGUMENT columns

Name	Type	Description
oid	oid	Row identifier (hidden column)
argument_position	integer	Location of a parameter of a scheduled task or program.
argument_type	name	Parameter type of a scheduled task or program.
job_name	text	Name of a scheduled task or program.
argument_name	text	Parameter name of a scheduled task or program. The scheduled task inherits the parameter name of the program. Therefore, this parameter is null.
argument_value	text	Parameter value of a scheduled task. (The program cannot bind a value.)
default_value	text	Default parameter value of a program.

12.2.14.6 GS_JOB_ATTRIBUTE

GS_JOB_ATTRIBUTE records attributes of DBE_SCHEDULER scheduled tasks, including basic attributes of scheduled tasks, scheduled task classes, certificates, authorization, programs, and schedules. Common users do not have the permission to access the newly installed database instance.

Table 12-53 GS_JOB_ATTRIBUTE columns

Name	Type	Description
oid	oid	Row identifier (hidden column)
job_name	text	Names of scheduled tasks, scheduled task classes, certificates, programs, and schedules, and authorized usernames.
attribute_name	text	Attribute names of scheduled tasks, scheduled task classes, certificates, programs, and schedules, and authorized content.
attribute_value	text	Attribute values of scheduled tasks, scheduled task classes, certificates, programs, and schedules.

12.2.14.7 GS_PACKAGE

GS_PACKAGE records package information.

Table 12-54 GS_PACKAGE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
pkgnamespace	oid	Schema to which a package belongs
pkgowner	oid	Owner of a package
pkgname	name	Name of a package
pkgspecsrc	text	Package specification
pkgbodydeclsrc	text	Package body
pkgbodyinitsrc	text	Package initialization source
pkgacl	aclitem[]	Access permission
pkgsecdef	boolean	Whether a user has the definer permission on the package.

12.2.14.8 GS_PLAN_TRACE

The GS_PLAN_TRACE system catalog stores plan traces. It records details about the plan generation process of DML statements. Only initial users have the write permission on this system catalog. Users with the sysadmin permission can read this system catalog.

Table 12-55 GS_PLAN_TRACE columns

Name	Type	Description
query_id	text	Unique ID of the current request
query	text	SQL statement of the current request. The value of this field cannot exceed the value of **track_activity_query_size**.
unique_sql_id	bigint	Unique ID of the SQL statement of the current request
plan	text	Query plan text corresponding to the current request SQL statement. The value of this field cannot exceed 10 KB.
plan_trace	text	Details about the query plan generation process corresponding to the SQL statement of the current request. The value of this field cannot exceed 300 MB.
owner	oid	OID of the user who initiates the current SQL request
modifydate	timestamp with time zone	Time when the current plan trace is updated (that is, time when the plan trace is created)

12.2.14.9 GS_POLICY_LABEL

GS_POLICY_LABEL records the resource label configuration information. One resource label corresponds to one or more records, and each record identifies the resource label to which a database resource belongs. Only the system administrator or security policy administrator can access this system catalog.

Fully Qualified Domain Name (FQDN) identifies an absolute path of a database resource.

Table 12-56 GS_POLICY_LABEL columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)
labelname	name	Resource label name
labeltype	name	Resource tag type. Currently, the value is **RESOURCE**.
fqdnnamespace	oid	OID of a namespace to which an identified database resource belongs

Name	Type	Description
fqdnid	oid	OID of an identified database resource. If the database resource is a column, this column is the OID of the catalog.
relcolumn	name	Column name. If the identified database resource is a column, this column indicates the column name. Otherwise, this column is empty.
fqdntype	name	Type of the identified database resource, for example, schema, table, column, or view

12.2.14.10 GS_RECYCLEBIN

GS_RECYCLEBIN describes details about objects in the recycle bin.

Table 12-57 GS_RECYCLEBIN columns

Name	Type	Description
oid	oid	System column.
rcybaseid	oid	Base table object ID, which references **gs_recyclebin.oid**.
rcydbid	oid	OID of the database to which the current object belongs.
rcyrelid	oid	OID of the current object.
rcyname	name	Name of the object in the recycle bin. The format is BIN*$unique_id$oid$*0. *unique_id* indicates the unique identifier with a maximum of 16 characters, and *oid* indicates the OID.
rcyoriginname	name	Original object name.
rcyoperation	char	Operation type. ● **d**: drop ● **t**: truncate

Name	Type	Description
rcytype	integer	Object type. ● **0**: table ● **1**: index. ● **2**: TOAST table. ● **3**: TOAST index. ● **4**: sequence, indicating the sequence object that is automatically associated with the serial, bigserial, smallserial, and largeserial types. ● **5**: partition. ● **6**: global index. ● **7**: materialized view
rcyrecyclecsn	bigint	CSN when an object is dropped or truncated.
rcyrecycletime	timestamp with time zone	Time when an object is dropped or truncated.
rcycreatecsn	bigint	CSN when an object is created.
rcychangecsn	bigint	CSN when an object definition is modified.
rcynamespace	oid	OID of the namespace that contains this relationship.
rcyowner	oid	Owner of the relationship.
rcytablespace	oid	Tablespace in which this relationship is stored. If the value is **0**, the default tablespace of the database is used. This column is meaningless if the relationship has no on-disk file.
rcyrelfilenode	oid	File name of the recycle bin object on a disk, or **0** if none, which is used to restore the physical file when the TRUNCATE object is restored.
rcycanrestore	Boolean	Specifies whether flashback can be performed separately.
rcycanpurge	Boolean	Specifies whether the purge operation can be performed independently.
rcyfrozenxid	xid32	All transaction IDs before this one have been replaced with a permanent ("frozen") transaction ID in the table.
rcyfrozenxid64	xid	All transaction IDs before this one have been replaced with a permanent ("frozen") transaction ID in the table.

Name	Type	Description
rcybucket	oid	Bucket information in 13.2.15.41 PG_HASHBUCKET.

12.2.14.11 GS_SECURITY_LABEL

The GS_SECURITY_LABEL is a shared system catalog that records information about security labels. Each record corresponds to a security label.

Table 12-58 GS_SECURITY_LABEL columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
label_name	name	Security label name.
label_content	text	Content of the security label.

12.2.14.12 GS_SQL_PATCH

GS_SQL_PATCH records the status information about all SQL patches.

Table 12-59 GS_SQL_PATCH columns

Name	Type	Description
patch_name	name	Patch name.
unique_sql_id	bigint	Global unique ID.
owner	oid	ID of the user who creates the patch.
enable	boolean	Specifies whether the patch takes effect.
status	"char"	Patch status (reserved column).
abort	boolean	Specifies whether the patch is an abort hint.
hint_string	text	Hint text.
hint_node	pg_node_tree	Hint parsing and serialization result.
original_query	text	Original statement (reserved column).
patched_query	text	Patched statement (reserved column).

Name	Type	Description
original_query_tree	pg_node_tree	Original statement parsing result (reserved column).
patched_query_tree	pg_node_tree	Patched statement parsing result (reserved column).
description	text	Patch description.
parent_unique_sql_id	bigint	Globally unique ID of the outer statement of the SQL statement for which the patch takes effect. The value of this parameter is **0** for statements outside a stored procedure. For statements inside the stored procedure, and the value of this parameter is the globally unique ID of the statement that invokes the stored procedure.

12.2.14.13 GS_STATISTIC_EXT_HISTORY

GS_STATISTIC_EXT_HISTORY is a multi-column historical statistics management table. It stores historical extended statistics about tables in the database, including multi-column statistics and expression statistics (supported later). You can specify the extended statistics to collect. This system catalog is accessible only to system administrators.

Table 12-60 GS_STATISTIC_EXT_HISTORY columns

Name	Type	Description
starelid	oid	Table or index that the described column belongs to.
starelkind	"char"	Type of the object to which a table belongs. **'c'** indicates an ordinary table, and **'p'** indicates a partitioned table.
stainherit	Boolean	Determines whether to collect statistics for objects that have inheritance relationship.
statimestamp	timestamp with time zone	Time when the statistics are collected.
stanullfrac	real	Percentage of column entries that are null.
stawidth	integer	Average stored width, in bytes, of non-null entries.

Name	Type	Description
stadistinct	real	Number of distinct, non-null data values in the column for database nodes. ● A value greater than 0 indicates the actual number of distinct values. ● A value less than 0 indicates the ratio of the **distinct** value to the total number of rows. For example, if **stadistinct** is **-0.5**, the actual **distinct** value is the total number of rows multiplied by 0.5. ● The value **0** indicates that the number of distinct values is unknown.
standistinct	real	Number of unique non-null data values in the **DN1** column. ● A value greater than 0 indicates the actual number of distinct values. ● A value less than 0 is the ratio of the **distinct** value to the total number of rows. For example, if **standistinct** is **-0.5**, the actual **distinct** value is the total number of rows multiplied by 0.5. ● The value **0** indicates that the number of distinct values is unknown.
standvfunc	"char"	Algorithm used to calculate the NDV based on the statistics. ● **d**: The original DUJ1 algorithm is used for estimation. ● **c**: The C19 algorithm is used for estimation.
staorigin	"char"	Source of the statistics collection mode. ● **a**: The collection is triggered by AUTOANALYZE. ● **m**: The collection is triggered by manual ANALYZE. ● **g**: The gsstat thread is triggered to perform ANALYZE collection when a large amount of data is inserted.
stakindN	smallint	Code number stating that the type of statistics is stored in slot *N* of the *pg_statistic* row. The value of *N* ranges from 1 to 5.
staopN	oid	Operator used to generate the statistics stored in slot *N*. For example, a histogram slot shows the < operator that defines the sort order of the data. The value of *N* ranges from 1 to 5.
stakey	int2vector	Array of a column ID.

Name	Type	Description
stanumbersN	real[]	Numerical statistics of the appropriate type for slot *N*. The value is **NULL** if the slot does not involve numerical values. The value of *N* ranges from 1 to 5.
stavaluesN	anyarray	Column data values of the appropriate type for slot *N*. The value is **NULL** if the slot type does not store any data values. Each array's element values are actually of the specific column's data type so there is no way to define these columns' type more specifically than anyarray. The value of *N* ranges from 1 to 5.
staexprs	pg_node_tree	Expression corresponding to the extended statistics information.
stasource	"char"	Source of extended statistics: ● **'a'**: indicates that the statistics data is automatically created, which is controlled by the GUC parameter **auto_statistic_ext_columns**. ● **'m'**: indicates that a user manually creates the statistics data using analyze tablename ((column list)) or alter table tablename add statistics ((column list)).
stastatus	"char"	Status of extended statistics: ● **'a'**: active and available. ● **'d'**: disabled. Related information is not collected, and the optimizer does not use the data when generating a plan. You can use the alter table tablename disable/enable statistics((column list)) syntax to modify the status of extended statistics.
staextname	name	Alias of the multi-column group of multi-column statistics.

12.2.14.14 GS_STATISTIC_HISTORY

GS_STATISTIC_HISTORY is a single-column historical statistics management table that stores historical statistics about tables and index columns in a database. By default, only the system administrator can access the system catalog. Common users can access the system catalog only after being authorized.

Table 12-61 GS_STATISTIC_HISTORY columns

Name	Type	Description
starelid	oid	Table or index that the described column belongs to.

Name	Type	Description
starelkind	"char"	Type of an object.
staattnum	smallint	Number of the described column in the table, starting from 1.
stainherit	Boolean	Determines whether to collect statistics for objects that have inheritance relationship.
statimestamp	timestamp with time zone	Time when the statistics are collected.
stanullfrac	real	Percentage of column entries that are null.
stawidth	integer	Average stored width, in bytes, of non-null entries.
stadistinct	real	Number of distinct, non-null data values in the column for database nodes. ● A value greater than 0 indicates the actual number of distinct values. ● A value less than 0 indicates the ratio of the **distinct** value to the total number of rows. For example, if **stadistinct** is **-0.5**, the actual **distinct** value is the total number of rows multiplied by 0.5. ● The value **0** indicates that the number of distinct values is unknown.
standvfunc	"char"	Algorithm used to calculate the NDV based on the statistics. ● **d**: The original DUJ1 algorithm is used for estimation. ● **c**: The C19 algorithm is used for estimation.
staorigin	"char"	Source of the statistics collection mode. ● **a**: The collection is triggered by AUTOANALYZE. ● **m**: The collection is triggered by manual ANALYZE. ● **g**: The gsstat thread is triggered to perform ANALYZE collection when a large amount of data is inserted.
stakindN	smallint	Code number stating that the type of statistics is stored in slot *N* of the *pg_statistic* row. The value of *N* ranges from 1 to 5.
staopN	oid	Operator used to generate the statistics stored in slot *N*. For example, a histogram slot shows the < operator that defines the sort order of the data. The value of *N* ranges from 1 to 5.

Name	Type	Description
stanumbersN	real[]	Numerical statistics of the appropriate type for slot *N*. The value is **NULL** if the slot does not involve numerical values. The value of *N* ranges from 1 to 5.
stavaluesN	anyarray	Column data values of the appropriate type for slot *N*. The value is **NULL** if the slot type does not store any data values. Each array's element values are actually of the specific column's data type so there is no way to define these columns' type more specifically than anyarray. The value of *N* ranges from 1 to 5.
stadndistinct	real	Number of unique non-null data values in the **DN1** column. ● A value greater than 0 indicates the actual number of distinct values. ● A value less than 0 is the ratio of the **distinct** value to the total number of rows. For example, if **stadndistinct** is **-0.5**, the actual **distinct** value is the total number of rows multiplied by 0.5. ● The value **0** indicates that the number of distinct values is unknown.
staextinfo	text	Information about extension statistics. Reserved column.

12.2.14.15 GS_TABLESTATS_HISTORY

GS_TABLESTATS_HISTORY is a table for managing historical statistics at the table, index, and partition levels. It stores historical statistics about tables, indexes, and partitions in a database.

Table 12-62 GS_TABLESTATS_HISTORY columns

Name	Type	Description
relid	oid	Unique identifier of a table, index, or partition in pg_class/pg_partition.
relname	name	Name of a table, index, or partition.
relnamespace	oid	OID of the namespace that contains this object.

Name	Type	Description
relkind	"char"	Object type. • **r**: ordinary table. • **I**: table-level index. • **i**: partitioned index. • **p**: level-1 partition. • **s**: level-2 partition.
reltimestamp	timestamp with time zone	Time when the statistics are collected.
relpages	double precision	Size of the table on disk in pages. This is only an estimate used by the optimizer.
reltuples	double precision	Number of rows in the table. This is only an estimate used by the optimizer.
relallvisible	integer	Number of pages marked as all visible in the table.

12.2.14.16 GS_TXN_SNAPSHOT

GS_TXN_SNAPSHOT is a timestamp-CSN mapping table. It periodically samples and maintains an appropriate time range to estimate the CSN value corresponding to the timestamp in the range.

Table 12-63 GS_TXN_SNAPSHOT columns

Name	Data Type	Description
snptime	timestamp with time zone	Snapshot time
snpxmin	bigint	Minimum snapshot ID
snpcsn	bigint	Snapshot CSN
snpsnapshot	text	Serialized snapshot text

12.2.14.17 GS_UID

GS_UID records the unique identification meta information of the hasuids attribute table in the database.

Table 12-64 GS_UID columns

Name	Type	Description
relid	oid	OID of a table.
uid_backup	bigint	Largest unique identifier that can be assigned to a table.

12.2.14.18 GS_WORKLOAD_RULE

The GS_WORKLOAD_RULE system catalog records information about SQL concurrency control rules. There is no permission restriction on this system catalog. All users can query this system catalog.

Table 12-65 GS_WORKLOAD_RULE columns

Name	Type	Description
rule_id	bigint	Concurrency control rule ID, which is automatically generated by the system.
rule_name	name	Name of the concurrency control rule, which is used for search. The name may not be unique and can be **NULL**.
databases	name[]	List of databases on which the concurrency control rules take effect. If the value is **NULL**, the concurrency control rules take effect for all databases.
max_workload	bigint	Maximum number of concurrent rule settings.
is_valid	boolean	Determines whether the concurrency control rules take effect. If the concurrency control rules time out, the value is set to **false**.
start_time	timestamp with time zone	Start time of the concurrency control rules. The value **NULL** indicates that the rules take effect from now on.
end_time	timestamp with time zone	End time of the concurrency control rules. The value **NULL** indicates that the rules are always effective.
rule_type	text	Concurrency control rule type. Currently, only **"sqlid"**, **"select"**, **"insert"**, **"update"**, **"delete"**, **"merge"**, and **"resource"** are supported. Other values are invalid.

Name	Type	Description
option_val	text[]	Parameter values of concurrency control rules, including SQL ID, keyword list, and resource restriction. For details, see the description of the **gs_add_workload_rule** API.
node_names	text[]	List of nodes on which the concurrency control rules take effect. This parameter is reserved and does not take effect currently.
user_names	text[]	List of users for which the concurrency control rules take effect. This parameter is reserved and does not take effect currently.

12.2.14.19 PG_AGGREGATE

PG_AGGREGATE records information about aggregate functions. Each entry in PG_AGGREGATE is an extension of an entry in PG_PROC. The PG_PROC entry carries the aggregate's name, input and output data types, and other information that is similar to ordinary functions.

Table 12-66 PG_AGGREGATE columns

Name	Type	Reference	Description
aggfnoid	regproc	**PG_PROC**.proname	**PG_PROC** proname of the aggregate function
aggtransfn	regproc	**PG_PROC**.proname	Transition function
aggcollectfn	regproc	**PG_PROC**.proname	Collect function
aggfinalfn	regproc	**PG_PROC**.proname	Final function (**0** if none)
aggsortop	oid	**PG_OPERATOR**.oid	Associated sort operator (**0** if none)
aggtranstype	oid	**PG_TYPE**.oid	Data type of the aggregate function's internal transition (state) data The possible values and their meanings are defined by the types in **pg_type.h**. The main two types are polymorphic (isPolymorphicType) and non-polymorphic.

Name	Type	Reference	Description
agginitval	text	-	Initial value of the transition state. This is a text column containing the initial value in its external string representation. If this column is null, the transition state value starts from null.
agginitcollect	text	-	Initial value of the collection state. This is a text column containing the initial value in its external string representation. If this column is null, the collection state value starts from null.
aggkind	"char"	-	Type of the aggregate function: • **n**: normal aggregate • **o**: ordered set aggregate
aggnumdirectargs	smallint	-	Number of direct parameters (non-aggregation-related parameters) of the aggregate function of the ordered set aggregate type. For an aggregate function of the normal aggregate type, the value is **0**.
agginitfn	regproc	PG_PROC.proname	Initialization function.

12.2.14.20 PG_AM

PG_AM records information about index access methods. There is one row for each index access method supported by the system.

Table 12-67 PG_AM columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
amname	name	-	Name of the access method
amstrategies	smallint	-	Number of operator strategies for the access method (**0** if the access method does not have a fixed set of operator strategies)

Name	Type	Reference	Description
amsupport	smallint	-	Number of support routines for the access method
amcanorder	Boolean	-	Whether the access method supports ordered scans sorted by the indexed column's value ● **t** (true): supported. ● **f** (false): not supported.
amcanorderbyop	Boolean	-	Whether the access method supports ordered scans sorted by the result of an operator on the indexed column ● **t** (true): supported. ● **f** (false): not supported.
amcanbackward	Boolean	-	Whether the access method supports backward scanning ● **t** (true): supported. ● **f** (false): not supported.
amcanunique	Boolean	-	Whether the access method supports unique indexes ● **t** (true): supported. ● **f** (false): not supported.
amcanmulticol	Boolean	-	Whether the access method supports composite indexes ● **t** (true): supported. ● **f** (false): not supported.
amoptionalkey	Boolean	-	Whether the access method supports scanning without any constraint for the first index column ● **t** (true): supported. ● **f** (false): not supported.
amsearcharray	Boolean	-	Whether the access method supports **ScalarArrayOpExpr** searches ● **t** (true): supported. ● **f** (false): not supported.
amsearchnulls	Boolean	-	Whether the access method supports **IS NULL/NOT NULL** searches ● **t** (true): supported. ● **f** (false): not supported.

Name	Type	Reference	Description
amstorage	Boolean	-	Whether the index storage data type can differ from the column data type • **t** (true): allowed. • **f** (false): not allowed.
amclusterable	Boolean	-	Whether an index of this type can be clustered on • **t** (true): allowed. • **f** (false): not allowed.
ampredlocks	Boolean	-	Whether an index of this type manages fine-grained predicate locks • **t** (true): allowed. • **f** (false): not allowed.
amkeytype	oid	OID in **PG_TYPE**	Type of data stored in index (**0** if it is not a fixed type)
aminsert	regproc	**PG_PROC**.proname	"Insert this tuple" function
ambeginscan	regproc	**PG_PROC**.proname	"Prepare for index scan" function
amgettuple	regproc	**PG_PROC**.proname	"Next valid tuple" function (**0** if none)
amgetbitmap	regproc	**PG_PROC**.proname	"Fetch all valid tuples" function (**0** if none)
amrescan	regproc	**PG_PROC**.proname	"(Re)start index scan" function
amendscan	regproc	**PG_PROC**.proname	"Clean up after index scan" function
ammarkpos	regproc	**PG_PROC**.proname	"Mark current scan position" function
amrestrpos	regproc	**PG_PROC**.proname	"Restore marked scan position" function
ammerge	regproc	**PG_PROC**.proname	"Merge multiple indexes" function
ambuild	regproc	**PG_PROC**.proname	"Build new index" function
ambuildempty	regproc	**PG_PROC**.proname	"Build empty index" function

Name	Type	Reference	Description
ambulkdelete	regproc	PG_PROC.proname	Bulk-delete function
amvacuumcleanup	regproc	PG_PROC.proname	Post-**VACUUM** cleanup function
amcanreturn	regproc	PG_PROC.proname	Function to check whether the index supports index-only scans (**0** if none)
amcostestimate	regproc	PG_PROC.proname	Function to estimate cost of an index scan
amoptions	regproc	PG_PROC.proname	Function to parse and validate **reloptions** for an index

12.2.14.21 PG_AMOP

PG_AMOP records information about operators associated with access method operator families. There is one row for each operator that is a member of an operator family. A family member can be either a search operator or an ordering operator. An operator can appear in more than one family, but cannot appear in more than one search position nor more than one ordering position within a family.

Table 12-68 PG_AMOP columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified).
amopfamily	oid	PG_OPFAMILY.oid	Operator family of this entry.
amoplefttype	oid	**oid** in PG_TYPE	Left-hand input data type of the operator. For details about the possible values and their descriptions, see 7.3 Data Types.
amoprighttype	oid	**oid** in PG_TYPE	Right-hand input data type of the operator For details about the possible values and their descriptions, see 7.3 Data Types.

Name	Type	Reference	Description
amopstrategy	smallint	-	Number of operator strategies.
amoppurpose	"char"	-	Purpose of the operator. • **s**: search. • **o**: order.
amopopr	oid	PG_OPERATOR.oid	OID of the operator.
amopmethod	oid	PG_AM.oid	Operator family of the index access method.
amopsortfamily	oid	PG_OPFAMILY.oid	The B-tree operator family according to which this entry sorts for an ordering operator (**0** for a search operator)

A search operator entry indicates that an index of this operator family can be searched to find all rows satisfying **WHERE indexed_column operator constant**. Obviously, such an operator must return a Boolean value, and its left-hand input type must match the index's column data type.

An ordering operator entry indicates that an index of this operator family can be scanned to return rows in the order represented by **ORDER BY indexed_column operator constant**. Such an operator could return any sortable data type, though again its left-hand input type must match the index's column data type. The exact semantics of **ORDER BY** are specified by the **amopsortfamily** column, which must reference the B-tree operator family for the operator's result type.

12.2.14.22 PG_AMPROC

PG_AMPROC records information about the support procedures associated with the access method operator families. There is one row for each support procedure that belongs to an operator family.

Table 12-69 PG_AMPROC columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
amprocfamily	oid	PG_OPFAMILY.oid	Operator family of this entry

Name	Type	Reference	Description
amproclefttype	oid	PG_TYPE.oid	Left-hand input data type of the associated operator For details about the possible values and their descriptions, see 8.3 Data Type.
amprocrighttype	oid	PG_TYPE.oid	Right-hand input data type of the associated operator For details about the possible values and their descriptions, see 8.3 Data Type.
amprocnum	smallint	-	Support procedure number
amproc	regproc	PG_PROC.proname	OID of the procedure

The usual interpretation of the **amproclefttype** and **amprocrighttype** columns is that they identify the left and right input types of the operator(s) that a particular support procedure supports. For some access methods, these match the input data type(s) of the support procedure itself; for others not. There is a notion of "default" support procedures for an index, which are those with **amproclefttype** and **amprocrighttype** both equal to the index opclass's **opcintype**.

12.2.14.23 PG_ATTRDEF

PG_ATTRDEF records default values of columns.

Table 12-70 PG_ATTRDEF columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
adrelid	oid	Table to which a column belongs.
adnum	smallint	Number of columns.
adbin	pg_node_tree	Internal representation of the default value of a column or of a generated expression.
adsrc	text	Readable representation of the default value of a column or of a generated expression.
adgencol	"char"	Specifies whether a column is a generated column. The value **s** indicates that the column is a generated column, and the value **'\0** indicates that the column is a common column. The default value is **'\0**.

Name	Type	Description
adbin_on_update	pg_node_tree	Internal representation of the attribute expression **on update current_timestamp** of the column.
adsrc_on_update	text	Internal representation of the readable attribute expression **on update current_timestamp**.

12.2.14.24 PG_ATTRIBUTE

PG_ATTRIBUTE records information about table columns.

Table 12-71 PG_ATTRIBUTE columns

Name	Type	Description
attrelid	oid	Table to which a column belongs.
attname	name	Column name.
atttypid	oid	Column type.
attstattarget	integer	Level of details of statistics collected for this column by ANALYZE. • The value **0** indicates that no statistics should be collected. • A negative value indicates that the system default statistic object is used. • The exact meaning of positive values is data type-dependent. For scalar data types, **attstattarget** is both the target number of "most common values" to collect, and the target number of histogram bins to create.
attlen	smallint	Copy of typlen in **13.2.15.77 PG_TYPE** of the field type.
attnum	smallint	Number of the column.
attndims	integer	Number of dimensions if the column is an array (**0** in other cases).
attcacheoff	integer	This column is always set to **–1** on disks. When it is loaded into a row descriptor in the memory, it may be updated to cache the offset of the columns in the row.

Name	Type	Description
atttypmod	integer	Type-specific data supplied at the table creation time (for example, the maximum length of a **varchar** column). This column is used as the third parameter when passing to type-specific input functions and length coercion functions. The value will generally be **–1** for types that do not need ATTTYPMOD.
attbyval	Boolean	Copy of typbyval in **13.2.15.77 PG_TYPE** of the column type.
attstorage	"char"	Copy of typstorage in **13.2.15.77 PG_TYPE** of the column type.
attalign	"char"	Copy of typalign in **13.2.15.77 PG_TYPE** of the column type.
attnotnull	Boolean	A non-null constraint. It is possible to change this column to enable or disable the constraint.
atthasdef	Boolean	This column has a default value, in which case there will be a corresponding entry in **13.2.15.24 PG_ATTRDEF** that actually defines the value.
attisdropped	Boolean	Indicates that this column has been deleted and is no longer valid. A deleted column is still physically present in the table but is ignored by the analyzer, so it cannot be accessed through SQL.
attislocal	Boolean	Indicates that this column is locally defined in the relationship. Note that a column can be locally defined and inherited simultaneously.
attcmprmode	tinyint	Compressed modes for a specific column. The compressed mode includes: • **0**: not compressed (**ATT_CMPR_NOCOMPRESS**) • **1**: DELTA compression algorithm (**ATT_CMPR_DELTA**) • **2**: dictionary compression algorithm (**ATT_CMPR_DICTIONARY**) • **3**: prefix compression algorithm (**ATT_CMPR_PREFIX**) • **4**: digital string compression algorithm (**ATT_CMPR_NUMSTR**)
attinhcount	integer	Number of direct ancestors that this column has. A column with an ancestor cannot be dropped nor renamed.
attcollation	oid	Defined collation of a column.
attacl	aclitem[]	Permissions for column-level access.

Name	Type	Description
attoptions	text[]	Column attribute. Currently, the following attributes are supported: **n_distinct**: number of **distinct** values of a column (excluding subtables). **n_distinct_inherited**: number of **distinct** values of a column (including subtables).
attfdwoptions	text[]	Column attribute of a foreign table. Currently, **dist_fdw**, **file_fdw**, and **log_fdw** do not use foreign table column attributes.
attinitdefval	bytea	Stores the default value expression. ADD COLUMN in the row-store table must use this column.
attkvtype	tinyint	Key value type for a column. Value: **0**: default value (**ATT_KV_UNDEFINED**) **1**: dimension (**ATT_KV_TAG**) **2**: indicator (**ATT_KV_FIELD**) **3**: time column (**ATT_KV_TIMETAG**)
attidentity	"char"	Identity type of a column. Value: **'0'** or **'\0'**: The column is a non-IDENTITY column. **'a'**: The **IDENTITY** column attribute is of the **ALWAYS** type. **'d'**: The **IDENTITY** column attribute is of the **BY DEFAULT** type. **'n'**: The **IDENTITY** column attribute is of the **BY DEFAULT ON NULL** type.

12.2.14.25 PG_CAST

PG_CAST records the conversion relationship between data types.

Table 12-72 PG_CAST columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
castsource	oid	OID of the source data type.
casttarget	oid	OID of the target data type.
castfunc	oid	OID of the conversion function. The value **0** indicates that no conversion function is required.

Name	Type	Description
castcontext	"char"	Conversion mode between the source and target data types. ● **'e'**: Only explicit conversion can be performed (using the CAST or :: syntax). ● **'i'**: Implicit conversion can be performed. ● **'a'**: Both explicit and implicit conversion can be performed between data types.
castmethod	"char"	Conversion method. ● **'f'**: Conversion is performed using the specified function in the **castfunc** column. ● **'b'**: Binary forcible conversion rather than the specified function in the **castfunc** column is performed between data types.

12.2.14.26 PG_CLASS

PG_CLASS records database objects and their relationship.

Table 12-73 PG_CLASS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
relname	name	Name of an object, such as a table, index, or view.
relnamespace	oid	OID of the namespace that contains the relationship.
reltype	oid	Data type that corresponds to the table's row type. The index is 0 because the index does not have PG_TYPE records.
reloftype	oid	OID of the composite type (**0** for other types).
relowner	oid	Owner of the relationship.
relam	oid	Access method used, such as B-tree, if the row is an index.
relfilenode	oid	Name of the on-disk file of this relationship (**0** if such file does not exist).
reltablespace	oid	Tablespace in which this relationship is stored. If the value is **0**, the default tablespace in this database is used. This column is meaningless if the relationship has no on-disk file.

Name	Type	Description
relpages	double precision	Size of the on-disk representation of the table in pages (of size **BLCKSZ**). This is only an estimate used by the optimizer.
reltuples	double precision	Number of rows in the table. This is only an estimate used by the optimizer.
relallvisible	integer	Number of pages marked as all visible in the table. This column is used by the optimizer for optimizing SQL execution. It is updated by VACUUM, ANALYZE, and a few DDL statements such as CREATE INDEX.
reltoastrelid	oid	OID of the TOAST table associated with the table (**0** if no TOAST table exists). The TOAST table stores large columns "offline" in a secondary table.
reltoastidxid	oid	OID of the index for a TOAST table (**0** for a table other than a TOAST table)
relhasindex	Boolean	Its value is **true** if this column is a table and has (or recently had) at least one index. It is set by CREATE INDEX but is not immediately cleared by DROP INDEX. If the VACUUM thread detects that a table has no index, it clears the **relhasindex** column and sets the value to **false**.
relisshared	Boolean	Its value is **true** if the table is shared across all database nodes in the database. Otherwise, the value is **false**. Only certain system catalogs (such as PG_DATABASE) are shared.
relpersistence	"char"	• **p**: permanent table • **u**: non-log table • **t**: temporary table • **g**: global temporary table.
relkind	"char"	• **r**: ordinary table • **i**: index • **I**: global index of a partitioned table • **s**: sequence • **L**: long sequence • **v**: view • **c**: composite type • **t**: TOAST table • **f**: foreign table • **m**: materialized view

Name	Type	Description
relnatts	smallint	Number of user columns in the relationship (excluding system columns). **PG_ATTRIBUTE** has the same number of rows as the user columns.
relchecks	smallint	Number of check constraints in the table. For details, see the system catalog **PG_CONSTRAINT**.
relhasoids	Boolean	Its value is **true** if an OID is generated for each row of the relationship. Otherwise, the value is **false**.
relhaspkey	Boolean	Its value is **true** if the table has (or once had) a primary key. Otherwise, the value is **false**.
relhasrules	Boolean	Its value is **true** if the table has rules. For details, see the system catalog **PG_REWRITE**.
relhastriggers	Boolean	The value is **true** if the table has (or once had) triggers. Triggers of the table and view are recorded in the system catalog **PG_TRIGGER**.
relhassubclass	Boolean	Its value is **true** if the table has (or once had) any inheritance child table. Otherwise, the value is **false**.
relcmprs	tinyint	Specifies whether the compression feature is enabled for the table. Note that only batch insertion triggers compression, so ordinary CRUD does not trigger compression. • **0**: Tables that do not support compression (primarily system catalogs, on which the compression attribute cannot be modified). • **1**: The compression feature of the table data is NOCOMPRESS or has no specified keyword. • **2**: The compression feature of the table data is COMPRESS.
relrowmovement	Boolean	Specifies whether row migration is allowed when the partitioned table is updated. • **true**: Row migration is allowed. • **false**: Row migration is not allowed.
parttype	"char"	Specifies whether the table or index has the property of a partitioned table. • **p**: The table or index has the property of a partitioned table. • **n**: The table or index does not have the property of a partitioned table. • **s**: The table is a level-2 partitioned table.

Name	Type	Description
relfrozenxid	xid32	All transaction IDs before this one have been replaced with a permanent ("frozen") transaction ID in the table. This column is used to track whether the table needs to be vacuumed to prevent transaction ID wraparound (or to allow PG_CLOG to be shrunk). The value is **0** (**InvalidTransactionId**) if the relationship is not a table. To ensure forward compatibility, this column is reserved. The **relfrozenxid64** column is added to record the information.
relacl	aclitem[]	Access permissions. For details about the ACLItem type, see **ACLItem Type**. The command output of the query is as follows: *user1*=*privs*/*user2* indicates that the permission granted by user 2 to user 1 is **privs**. =*privs*/*user3* indicates that the permission granted to the **public** role by user 3 is **privs**. In the preceding command, user 1, user 2, and user 3 are the existing users or roles in the database, and **privs** indicates the permissions supported by the database. For details on permission descriptions, see **Table 12-74**.
reloptions	text[]	Table or index access method, using character strings in the format of "keyword=value"
relreplident	"char"	Identifier of a decoding column in logical decoding. • **d**: default (primary key, if any) • **n**: none • **f**: all columns • **i**: The indisreplident of the index is specified or the default index is used.
relfrozenxid64	xid	All transaction IDs before this one have been replaced with a permanent ("frozen") transaction ID in the table. This column is used to track whether the table needs to be vacuumed to prevent transaction ID wraparound (or to allow PG_CLOG to be shrunk). The value is **0** (**InvalidTransactionId**) if the relationship is not a table. For a global temporary table, this field is meaningless. You can view **relfrozenxid64** of the global temporary table of each session in the **pg_catalog.pg_gtt_relstats** view.

Name	Type	Description
relbucket	oid	Specifies whether the current catalog contains hash bucket shards. A valid OID points to the specific shard information recorded in the PG_HASHBUCKET catalog. **NULL** indicates that hash bucket shards are not included.
relbucketkey	int2vector	Hash partition column information. **NULL** indicates that the column information is not included.
relminmxid	xid	All multi-transaction IDs before this one have been replaced with a transaction ID in the table. This is used to track whether the table needs to be vacuumed in order to prevent multi-transaction IDs wraparound or to allow PG_CLOG to be shrunk. The value is **0** (**InvalidTransactionId**) if the relationship is not a table.

Table 12-74 Description of permissions

Parameter	Description
r	SELECT (read)
w	UPDATE (write)
a	INSERT (insert)
d	DELETE
D	TRUNCATE
x	REFERENCES
t	TRIGGER
X	EXECUTE
U	USAGE
C	CREATE
c	CONNECT
T	TEMPORARY
A	ALTER
P	DROP
m	COMMENT
i	INDEX
v	VACUUM

Parameter	Description
*	Authorization options for preceding permissions

12.2.14.27 PG_COLLATION

PG_COLLATION describes available collations, which are essentially mappings from an SQL name to operating system locale categories.

Table 12-75 PG_COLLATION columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
collname	name	-	Collation name (unique per namespace and encoding)
collnamespace	oid	OID in **PG_NAMESPACE**	OID of the namespace that contains this collation
collowner	oid	OID in **PG_AUTHID**	Owner of the collation
collencoding	integer	-	Encoding in which the collation is applicable, or **–1** if it works for any encoding. It is compatible with PostgreSQL.
collcollate	name	-	**LC_COLLATE** for this collation object
collctype	name	-	**LC_CTYPE** for this collation object
collpadattr	text	-	Collation padding attribute. • **NULL**: not applicable. • **NO PAD**: no padding. • **PAD SPACE**: blank spaces padded at the end.
collisdef	Boolean	-	Determines whether the collation is the default collation of the character set.

12.2.14.28 PG_CONSTRAINT

PG_CONSTRAINT records check, primary key, and unique constraints on tables.

Table 12-76 PG_CONSTRAINT columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
conname	name	Constraint name (not necessarily unique).
connamespace	oid	OID of the namespace that contains the constraint.
contype	"char"	• **c**: check constraint. • **p**: primary key constraint. • **u**: unique constraint. • **t**: trigger constraint. • **x**: mutual exclusion constraint. • **f**: foreign key constraint. • **s**: clustering constraint. • **i**: invalid constraint.
condeferrable	Boolean	Specifies whether the constraint is deferrable. • **true**: yes. • **false**: no.
condeferred	Boolean	Specifies whether the constraint can be deferrable by default. • **true**: yes. • **false**: no.
convalidated	Boolean	Specifies whether the constraint is valid. Currently, it can be set to **false** only for foreign key and check constraints. • **true**: valid. • **false**: invalid.
conrelid	oid	Table containing this constraint (**0** if it is not a table constraint).
contypid	oid	Domain containing this constraint (**0** if it is not a domain constraint).
conindid	oid	ID of the index associated with the constraint.
confrelid	oid	Referenced table if this constraint is a foreign key. Otherwise, the value is **0**.

Name	Type	Description
confupdtype	"char"	Foreign key update action code. ● **a**: no action. ● **r**: restriction. ● **c**: cascading. ● **n**: The parameter is set to **null**. ● **d**: The default value is used.
confdeltype	"char"	Foreign key deletion action code. ● **a**: no action. ● **r**: restriction. ● **c**: cascading. ● **n**: The parameter is set to **null**. ● **d**: The default value is used.
confmatchtype	"char"	Foreign key match type. ● **f**: full match. ● **p**: partial match. ● **u**: unspecified. (The NULL value can be matched if **f** is specified.)
conislocal	Boolean	Specifies whether the constraint is defined locally for the relationship. ● **true**: yes. ● **false**: no.
coninhcount	integer	Number of direct inheritance parent tables that this constraint has. When the value is not **0**, the constraint cannot be deleted or renamed.
connoinherit	Boolean	Specifies whether the constraint can be inherited. ● **true**: yes. ● **false**: no.
consoft	Boolean	Specifies whether the column indicates an informational constraint. ● **true**: yes. ● **false**: no.
conopt	Boolean	Specifies whether you can use the informational constraint to optimize the execution plan. ● **true**: yes. ● **false**: no.

Name	Type	Description
conkey	smallint[]	Column list of the constrained control if this column is a table constraint.
confkey	smallint[]	List of referenced columns if this column is a foreign key.
conpfeqop	oid[]	ID list of the equality operators for PK = FK comparisons if this column is a foreign key.
conppeqop	oid[]	ID list of the equality operators for PK = PK comparisons if this column is a foreign key.
conffeqop	oid[]	ID list of the equality operators for FK = FK comparisons if this column is a foreign key. The value is empty because foreign keys are not supported currently.
conexclop	oid[]	ID list of the per-column exclusion operators if this column is an exclusion constraint.
conbin	pg_node_tree	Internal representation of the expression if this column is a check constraint.
consrc	text	Readable representation of the expression if this column is a check constraint.
conincluding	smallint[]	Not for constraint, but will be included in the attribute column of **INDEX**.

NOTICE

- **consrc** is not updated when referenced objects change and does not track new column names. Instead of relying on this column to update, you are advised to use pg_get_constraintdef() to extract the definition of a check constraint.
- **relchecks** of **PG_CLASS** must agree with the number of check-constraint entries found in the table for each relationship.

12.2.14.29 PG_CONVERSION

PG_CONVERSION describes encoding conversion information.

Table 12-77 PG_CONVERSION columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)

Name	Type	Reference	Description
conname	name	-	Conversion name (unique within a namespace)
connamespace	oid	PG_NAMESPACE.oid	OID of the namespace that contains this conversion
conowner	oid	PG_AUTHID.oid	Owner of the conversion
conforencoding	integer	-	Source encoding ID
contoencoding	integer	-	Destination encoding ID
conproc	regproc	PG_PROC.proname	Conversion procedure
condefault	Boolean	-	If this is the default conversion, the value is **true**. Otherwise, the value is **false**.

12.2.14.30 PG_DATABASE

PG_DATABASE records information about available databases.

Table 12-78 PG_DATABASE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
datname	name	Database name
datdba	oid	Owner of the database, usually the user who created it
encoding	integer	Character encoding for the database
datcollate	name	Sequence used by the database
datctype	name	Character type used by the database
datistemplate	Boolean	Whether the database can be used as a template database
datallowconn	Boolean	If the value is **true**, users can connect to the database. If the value is **false**, no one can connect to this database. This column is used to protect the **template0** database from being altered.
datconnlimit	integer	Maximum number of concurrent connections allowed on this database. The value **-1** indicates no limit.

Name	Type	Description
datlastsysoid	oid	Last system OID in the database
datfrozenxid	xid32	Tracks whether the database needs to be vacuumed to prevent transaction ID wraparound. This column is discarded in the current version. To ensure forward compatibility, this column is reserved. The **datfrozenxid64** column is added to record the information.
dattablespace	oid	Default tablespace of the database
datcompatibility	name	Database compatibility mode. Currently, four compatibility modes are supported: A, B, C, and PG, indicating that the Oracle, MySQL, Teradata, and Postgres databases are compatible.
datacl	aclitem[]	Access permission
datfrozenxid64	xid	Tracks whether the database needs to be vacuumed to prevent transaction ID wraparound.
datminmxid	xid	All multi-transaction IDs before this one have been replaced with a transaction ID in the database. This is used to track whether the database needs to be vacuumed in order to prevent transaction IDs wraparound or to allow PG_CLOG to be shrunk. It is the minimum value of **relminmxid** in **PG_CLASS** of all tables in the database.
dattimezone	name	Database time zone. The default is PRC.

12.2.14.31 PG_DEPEND

PG_DEPEND records the dependency between database objects. This information allows **DROP** commands to find which other objects must be dropped by **DROP CASCADE** or prevent dropping in the **DROP RESTRICT** case.

See also **PG_SHDEPEND**, which performs a similar function for dependencies involving objects that are shared across databases.

Table 12-79 PG_DEPEND columns

Name	Type	Reference	Description
classid	oid	OID in **PG_CLASS**	OID of the system catalog where a dependent object resides

Name	Type	Reference	Description
objid	oid	Any OID column	OID of the dependent object
objsubid	integer	-	For a table column, this is the column number (**objid** and **classid** refer to the table itself). The value is **0** for all other object types.
refclassid	oid	OID in **PG_CLASS**	OID of the system catalog where a referenced object resides
refobjid	oid	Any OID column	OID of the referenced object
refobjsubid	integer	-	For a table column, this is the column number (**refobjid** and **refclassid** refer to the table itself). The value is **0** for all other object types.
deptype	"char"	-	A code defining the specific semantics of this dependency

In all cases, a PG_DEPEND entry indicates that the referenced object cannot be dropped without also dropping the dependent object. However, there are several subflavors identified by **deptype**:

- DEPENDENCY_NORMAL (n): A normal relationship between separately created objects. The dependent object can be dropped without affecting the referenced object. The referenced object can only be dropped by specifying CASCADE, in which case the dependent object is dropped too. Example: a table column has a normal dependency on its data type.
- DEPENDENCY_AUTO (a): The dependent object can be dropped separately from the referenced object, and should be automatically dropped (regardless of RESTRICT or CASCADE mode) if the referenced object is dropped. Example: a named constraint on a table is made autodependent on the table, so that it will go away if the table is dropped.
- DEPENDENCY_INTERNAL (i): The dependent object was created as part of creation of the referenced object, and is only a part of its internal implementation. A DROP of the dependent object will be disallowed outright (We'll tell the user to issue a DROP against the referenced object, instead). A DROP of the referenced object will be propagated through to drop the dependent object whether CASCADE is specified or not.
- DEPENDENCY_EXTENSION (e): The dependent object is a member of the extension of the referenced object (see **PG_EXTENSION**). The dependent object can be dropped only via DROP EXTENSION on the referenced object. Functionally this dependency type acts the same as an internal dependency, but it is kept separate for clarity and to simplify GS_DUMP.

NOTICE

The extended function is for internal use only. You are advised not to use it.

- DEPENDENCY_PIN (p): There is no dependent object; this type of entry is a signal that the system itself depends on the referenced object, and so that object must never be deleted. Entries of this type are created only by **initdb**. The columns for the dependent object contain zeroes.

12.2.14.32 PG_DESCRIPTION

PG_DESCRIPTION records optional descriptions (comments) for each database object. Descriptions of many built-in system objects are provided in the initial contents of PG_DESCRIPTION.

See also **PG_SHDESCRIPTION**, which provides a similar function for descriptions involving objects that are shared within the entire database.

Table 12-80 PG_DESCRIPTION columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object that this description pertains to
classoid	oid	PG_CLASS.oid	OID of the system catalog where the object appears
objsubid	integer	-	Column number for a comment on a table column (**objoid** and **classoid** refer to the table itself); **0** for all other object types
description	text	-	Arbitrary text that serves as the description of the object

12.2.14.33 PG_DIRECTORY

PG_DIRECTORY stores directory objects added by users. You can execute the **CREATE DIRECTORY** statement to add records to this system catalog. When **enable_access_server_directory** is set to **off**, only the initial user can create directory objects. When **enable_access_server_directory** is set to **on**, users with the SYSADMIN permission and users inheriting the built-in role permission gs_role_directory_create can create directory objects. Common users can access this system catalog only after being authorized.

Table 12-81 PG_DIRECTORY columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified)

Name	Type	Description
dirname	name	Name of a directory object
owner	oid	Owner of a directory object
dirpath	text	Directory path.
diracl	aclitem[]	Access permissions.

12.2.14.34 PG_ENUM

PG_ENUM contains entries showing the values and labels for each enumerated type. The internal representation of a given enumerated value is actually the OID of its associated row in PG_ENUM.

Table 12-82 PG_ENUM columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
enumtypid	oid	PG_TYPE.oid	OID of the PG_TYPE entry owning this enumerated value
enumsortorder	real	-	Sort position of this enumerated value within its enumerated type
enumlabel	name	-	Textual label for this enumerated value

The OIDs for PG_ENUM rows follow a special rule: even-numbered OIDs are guaranteed to be ordered in the same way as the sort ordering of their enumerated type. If two even OIDs belong to the same enumerated type, the smaller OID must have the smaller **enumsortorder** value. Odd-numbered OID values need bear no relationship to the sort order. This rule allows the enumerated comparison routines to avoid catalog lookups in many common cases. The routines that create and alter enumerated types attempt to assign even OIDs to enumerated values whenever possible.

When an enumerated type is created, its members are assigned sort-order positions from 1 to *n*. However, members added later might be given negative or fractional values of **enumsortorder**. The only requirement on these values is that they be correctly ordered and unique within each enumerated type.

12.2.14.35 PG_EXTENSION

PG_EXTENSION records information about the installed extensions. The default GaussDB extensions are PLPGSQL, DIST_FDW, FILE_FDW, LOG_FDW, DBLINK_FDW, PACKAGES, SECURITY_PLUGIN, GSSTAT_PLUGIN, NUMERIC_ENHANCE, PKG_DBE_RAW, PKG_DBE_OUTPUT, PKG_DBE_UTILITY, PKG_ILM,

PKG_DBE_XMLGEN, PKG_DBE_STATS, PKG_DBE_XML, PKG_DBE_DESCRIBE, and PKG_PL_PROFILER. This system catalog is for internal use only. You are advised not to use it.

Table 12-83 PG_EXTENSION

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
extname	name	Extension name.
extowner	oid	Owner of the extension.
extnamespace	oid	Namespace containing the extension's exported objects.
extrelocatable	Boolean	Specifies whether the extension can be relocated to another namespace. The value **true** indicates that relocation is allowed.
extversion	text	Version number of the extension.
extconfig	oid[]	Configuration information about the extension.
extcondition	text[]	Filter conditions for the extension's configuration information.

12.2.14.36 PG_FOREIGN_DATA_WRAPPER

PG_FOREIGN_DATA_WRAPPER records foreign-data wrapper definitions. A foreign-data wrapper is the mechanism by which external data, residing on foreign servers, is accessed.

Table 12-84 PG_FOREIGN_DATA_WRAPPER columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
fdwname	name	-	Name of a foreign-data wrapper
fdwowner	oid	OID in **PG_AUTHID**	Owner of a foreign-data wrapper
fdwhandler	oid	OID in **PG_PROC**	References a handler function that is responsible for supplying execution routines for a foreign-data wrapper The value is **0** if no handler is provided.

Name	Type	Reference	Description
fdwvalidator	oid	OID in PG_PROC	References a validator function that is responsible for checking the validity of the options given to a foreign-data wrapper, as well as options for foreign servers and user mappings using the foreign-data wrapper. The value is **0** if no validator is provided.
fdwacl	aclitem[]	-	Access permissions
fdwoptions	text[]	-	Foreign-data wrapper specific option, expressed in a string in the format of keyword=value

12.2.14.37 PG_FOREIGN_SERVER

PG_FOREIGN_SERVER records foreign server definitions. A foreign server describes a source of external data, such as a remote server. Foreign servers are accessed via foreign-data wrappers.

Table 12-85 PG_FOREIGN_SERVER columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
srvname	name	-	Name of a foreign server
srvowner	oid	OID in PG_AUTHID	Owner of the foreign server
srvfdw	oid	OID in PG_FOREIGN_DATA_WRAPPER	OID of the foreign-data wrapper on a foreign server
srvtype	text	-	Type of the server (optional)
srvversion	text	-	Version of the server (optional)
srvacl	aclitem[]	-	Access permissions
srvoptions	text[]	-	Option used for foreign servers, expressed in a string in the format of keyword=value

12.2.14.38 PG_HASHBUCKET

PG_HASHBUCKET records hash bucket information. It is an empty table and is not supported in the current version.

Table 12-86 PG_HASHBUCKET columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
bucketid	oid	This parameter is not supported in the current version. The value is **NULL**.
bucketcnt	integer	This parameter is not supported in the current version. The value is **NULL**.
bucketmapsize	integer	This parameter is not supported in the current version. The value is **NULL**.
bucketref	integer	This parameter is not supported in the current version. The value is **NULL**.
bucketvector	oidvector_extend	This parameter is not supported in the current version. The value is **NULL**.
bucketmap	text	This parameter is not supported in the current version. The value is **NULL**.
bucketversion	oidvector_extend	This parameter is not supported in the current version. The value is **NULL**.
bucketcsn	text	This parameter is not supported in the current version. The value is **NULL**.
bucketxid	text	This parameter is not supported in the current version. The value is **NULL**.

12.2.14.39 PG_INDEX

PG_INDEX records part of index information. The rest is mostly recorded in PG_CLASS.

Table 12-87 PG_INDEX columns

Name	Type	Description
indexrelid	oid	OID of the **PG_CLASS** entry for the index.
indrelid	oid	OID of the **PG_CLASS** entry for the table that uses the index.
indnatts	smallint	Number of columns in the index.

Name	Type	Description
indisunique	Boolean	Specifies whether the index is unique. ● **true**: The index is unique. ● **false**: The index is not unique.
indisprimary	Boolean	Specifies whether the index is the primary key of the table. ● **true**: The index is the primary key of the table. **indisunique** should always be **true** when the value of this column is **true**. ● **false**: The index is not the primary key of the table.
indisexclusion	Boolean	Specifies whether the index supports exclusive constraints. ● **true**: The index supports exclusive constraints. ● **false**: The index does not support exclusive constraints.
indimmediate	Boolean	Specifies whether to check the uniqueness of data to be inserted. ● **true**: The uniqueness check is performed immediately when data is inserted. ● **false**: The uniqueness check is not performed when data is inserted.
indisclustered	Boolean	Specifies whether the table is clustered on the index. ● **true**: The table is clustered on the index. ● **false**: The table is not clustered on the index.
indisusable	Boolean	Specifies whether the index is available for insert and select operations. ● **true**: The index is available for insert and select operations. ● **false**: The index is unavailable for insert and select operations.
indisvalid	Boolean	● **true**: The index can be used for query. ● **false**: The index is possibly incomplete and must still be modified by INSERT or UPDATE operations, but it cannot be securely used for queries. If it is a unique index, the uniqueness property is also not **true**.

Name	Type	Description
indcheckxmin	Boolean	• **true**: Queries must not use the index until the xmin of this row in PG_INDEX is lower than their **TransactionXmin**, because the table may contain broken HOT chains with incompatible rows that they can see. • **false**: Indexes can be used for query.
indisready	Boolean	• **true**: The index is available for inserting data. • **false**: The index is ignored when data is inserted or modified.
indkey	int2vector	This is an array of **indnatts** values indicating that this index creates table columns. For example, a value of **1 3** indicates that the first and the third columns make up the index key. The value **0** in this array indicates that the corresponding index attribute is an expression over the table columns, rather than a simple column reference.
indcollation	oidvector	OID of the collation corresponding to each index column. For details, see **PG_COLLATION**.
indclass	oidvector	For each column in the index key, this contains the OID of the operator class to use. For details, see section **12.2.15.53 PG_OPCLASS**.
indoption	int2vector	Array of values that store per-column flag bits. The meaning of the bits is defined by the index's access method.
indexprs	pg_node_tree	Expression trees (in **nodeToString()** representation) for index attributes that are not simple column references. It is a list with one element for each zero entry in indkey. The value is null if all index attributes are simple references.
indpred	pg_node_tree	Expression tree (in **nodeToString()** representation) for partial index predicate. If the index is not a partial index, this column is an empty string.
indisreplident	Boolean	Specifies whether the column of this index is a decoded column of logical decoding. • **true**: The column of this index is a decoded column of logical decoding. • **false**: The column of this index is not a decoded column of logical decoding.
indnkeyatts	smallint	Total number of columns in the index. The columns that exceed the value of **indnatts** are not involved in the index query.

Name	Type	Description
indcctmpid	oid	OID of the temporary table when the Ustore builds indexes online.
indisvisible	Boolean	• **true**: The index status is visible, that is, the optimizer can use the index. • **false**: The index is invisible. If the value of **enable_invisible_indexes** is **off**, the optimizer cannot use the index. If the value of **enable_invisible_indexes** is **on**, the optimizer can use the index.

12.2.14.40 PG_INHERITS

PG_INHERITS records information about table inheritance hierarchies. There is one entry for each direct child table in the database. Indirect inheritance can be determined by following chains of entries.

Table 12-88 PG_INHERITS columns

Name	Type	Reference	Description
inhrelid	oid	PG_CLASS.oid	OID of a child table
inhparent	oid	PG_CLASS.oid	OID of a parent table
inhseqno	integer	-	If there is more than one direct parent for a child table (multiple inheritances), this number tells the order in which the inherited columns are to be arranged. The count starts at 1.

12.2.14.41 PG_JOB

PG_JOB records detailed information about jobs created by users. Dedicated threads poll the system catalog PG_JOB and trigger jobs based on scheduled job execution time, and update job status in PG_JOB. This system catalog belongs to the Shared Relation category. All job records are visible to all databases. Common users can access this system catalog only after being authorized.

Table 12-89 PG_JOB columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).

Name	Type	Description
job_id	bigint	Job ID, which is the primary key and is unique (with a unique index).
current_postgres_pid	bigint	If the current job has been executed, the thread ID of this job is recorded. The default value is **–1**, indicating that the job has not yet been executed.
log_user	name	Username of the creator.
priv_user	name	Username of the job executor.
dbname	name	Name of the database in which the job will be executed.
node_name	name	Primary database node on which the job will be executed.
job_status	"char"	Execution status of the current task. The default value is **'s'**. The options are as follows: ● **'r'**: running ● **'s'**: successfully finished ● **'f'**: job failed ● **'d'**: disable If a job fails to be executed for 16 consecutive times, **job_status** is automatically set to **'d'**, and no more attempt will be made on this job. Note: When you disable a scheduled job (by setting **job_queue_processes** to **0**), the thread that monitors the job execution is not started, and the job status will not be updated. You can ignore this status. Only when the scheduled job function is enabled (**job_queue_processes** is not set to **0**), the system updates the value of this column based on the real-time job status.
start_date	timestamp without time zone	Start time of the first job execution, accurate to millisecond.
next_run_date	timestamp without time zone	Time when a scheduled job is executed next time. The time is accurate to milliseconds.
failure_count	smallint	Number of times the job has started and failed. If a job fails to be executed for 16 consecutive times, no more attempt will be made on it.
interval	text	Job execution interval.
last_start_date	timestamp without time zone	Start time of the last job execution, accurate to millisecond.

Name	Type	Description
last_end_date	timestamp without time zone	End time of the last job execution, accurate to millisecond.
last_suc_date	timestamp without time zone	Start time of the last successful job execution, accurate to millisecond.
this_run_date	timestamp without time zone	Start time of the ongoing job execution, accurate to millisecond.
nspname	name	Name of the schema used for job execution.
job_name	text	Name of the DBE_SCHEDULER scheduled job.
end_date	timestamp without time zone	Expiration time of the DBE_SCHEDULER scheduled job, accurate to millisecond.
enable	Boolean	Enabling status of the DBE_SCHEDULER scheduled job. The options are as follows: • **true**: enabled • **false**: disabled
failure_msg	text	Error information about the latest task execution.

12.2.14.42 PG_JOB_PROC

PG_JOB_PROC records the content of each job in the **PG_JOB** system catalog, including the PL/SQL code blocks and anonymous blocks. Storing such information in the system catalog PG_JOB and loading it to the shared memory will result in excessive memory usage. Therefore, such information is stored in a separate table and is retrieved when needed. Common users can access this system catalog only after being authorized.

Table 12-90 PG_JOB_PROC columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
job_id	integer	Foreign key, which is associated with **job_id** in **PG_JOB**.
what	text	Job content, which is the program content in the DBE_SCHEDULER scheduled job.

Name	Type	Description
job_name	text	Name of the DBE_SCHEDULER scheduled job or program.

12.2.14.43 PG_LANGUAGE

PG_LANGUAGE registers programming languages. You can use them and interfaces to write functions or stored procedures.

Table 12-91 PG_LANGUAGE columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
lanname	name	-	Language name
lanowner	oid	OID in PG_AUTHID	Owner of the language
lanispl	Boolean	-	• **true**: user-defined language. • **false**: internal language, for example, SQL. Currently, **gs_dump** still uses this column to determine which languages need to be dumped, but this might be replaced by a different mechanism in the future.
lanpltrusted	Boolean	-	• **true**: The language is trusted, which means that it is believed not to grant access to anything outside the normal SQL execution environment. • **false**: The language is untrusted. Only the initial user can create functions in untrusted languages.
lanplcallfoid	oid	OID in PG_PROC	For non-internal languages, this column references the language handler, which is a special function responsible for executing all functions that are written in the particular language.
laninline	oid	PG_PROC.oid	This column references a function responsible for executing "inline" anonymous code blocks (DO blocks). The value is **0** if inline blocks are not supported.

Name	Type	Reference	Description
lanvalidator	oid	PG_PROC.oid	This column references a language validator function responsible for checking the syntax and validity of new functions when they are created. The value is **0** if no validator is provided.
lanacl	aclitem[]	-	Access permissions

12.2.14.44 PG_LARGEOBJECT

PG_LARGEOBJECT records data making up large objects. A large object is identified by an OID assigned when it is created. Each large object is broken into segments or "pages" small enough to be conveniently stored as rows in PG_LARGEOBJECT. The amount of data per page is defined as **LOBLKSIZE**.

This system catalog is accessible only to system administrators.

Table 12-92 PG_LARGEOBJECT columns

Name	Type	Reference	Description
loid	oid	OID in PG_LARGEOBJECT_METADATA	Identifier of the large object that includes this page.
pageno	integer	-	Page number of this page within its large object (counting from zero).
data	bytea	-	Data stored in the large object. This will never be more than **LOBLKSIZE** bytes and might be less.

Each row of **PG_LARGEOBJECT** holds data for one page of a large object, beginning at byte offset (**pageno * LOBLKSIZE**) within the object. The implementation allows sparse storage: pages might be missing, and might be shorter than **LOBLKSIZE** bytes even if they are not the last page of the object. Missing regions within a large object read as zeroes.

12.2.14.45 PG_LARGEOBJECT_METADATA

PG_LARGEOBJECT_METADATA records metadata associated with large objects. The actual large object data is stored in **PG_LARGEOBJECT**.

Table 12-93 PG_LARGEOBJECT_METADATA columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
lomowner	oid	PG_AUTHID.oid	Owner of the large object
lomacl	aclitem[]	-	Access permissions

12.2.14.46 PG_NAMESPACE

PG_NAMESPACE records namespaces, that is, schema-related information. If the database object isolation attribute is enabled, users can view only the schema information that they have the permission to access.

Table 12-94 PG_NAMESPACE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
nspname	name	Name of a namespace
nspowner	oid	Owner of a namespace
nsptimeline	bigint	Timeline when a namespace is created on the database node. This column is for internal use and valid only on the database node.
nspacl	aclitem[]	Access permission For details, see 7.13.12.1 GRANT and 7.13.17.9 REVOKE.
in_redistribution	"char"	Specifies whether the content is in the redistribution state.
nspblockchain	Boolean	Specifies whether the mode is the tamper-proof mode. • **true**: The mode is the tamper-proof mode. • **false**: The mode is not the tamper-proof mode.
nspcollation	oid	Default collation of the namespace (a value may exist only when **sql_compatibility** is set to **'b'**).

12.2.14.47 PG_OBJECT

PG_OBJECT records the creator, creation time, and last modification time of objects of specified types (ordinary tables, indexes, sequences, views, stored procedures, and functions).

Table 12-95 PG_OBJECT columns

Name	Type	Description
object_oid	oid	Object identifier.
object_type	"char"	Object type. ● **r**: ordinary table ● **i**: index ● **s**: sequence ● **v**: view ● **P**: stored procedure and function ● **S**: package header ● **B**: package body
creator	oid	ID of a creator.
ctime	timestamp with time zone	Creation time of an object.
mtime	timestamp with time zone	Last modification time of an object. The modification operations include **ALTER**, **GRANT**, and **REVOKE**.
createcsn	bigint	CSN when an object is created.
changecsn	bigint	CSN when DDL operations are performed on a table or an index.
valid	Boolean	Validity of an object. **t** indicates valid, and **f** indicates invalid.

NOTICE

- Objects created or modified during database initialization (initdb) cannot be recorded. **PG_OBJECT** does not contain these object records.
- When an object created before the upgrade is modified again, the modification time (specified by **mtime**) is recorded. When DDL operations are performed on a table or an index, the transaction commit sequence number (specified by **changecsn**) of the transaction to which the table or index belongs is recorded. Because the creation time of the object cannot be obtained, **ctime** and **createcsn** are empty.
- The time recorded by **ctime** and **mtime** is the start time of the transaction to which the current operation belongs.
- The time of object modification due to capacity expansion is also recorded.
- **createcsn** and **changecsn** record the transaction commit sequence number of the transaction to which the current operation belongs.
- When **enable_gtt_concurrent_truncate** is set to **on**, the **mtime** column is not updated when the global temporary table is truncated.
- If the statement for creating an object has an undefined object, or the referenced object is modified or deleted, the object to be created will be invalid.

12.2.14.48 PG_OPCLASS

PG_OPCLASS defines index access method operator classes.

Each operator class defines semantics for index columns of a particular data type and a particular index access method. An operator class essentially specifies that a particular operator family is applicable to a particular indexable column data type. The set of operators from the family that are actually usable with the indexed column are data types that accept the left-hand column.

Table 12-96 PG_OPCLASS columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
opcmethod	oid	OID in **PG_AM**	Index access method operator class served by an operator class
opcname	name	-	Name of the operator class
opcnamespace	oid	OID in **PG_NAMESPACE**	Namespace of the operator class
opcowner	oid	OID in **PG_AUTHID**	Owner of the operator class
opcfamily	oid	OID in **PG_OPFAMILY**	Operator family containing the operator class
opcintype	oid	OID in **PG_TYPE**	Data type that the operator class indexes

Name	Type	Reference	Description
opcdefault	boolean	-	The value is **t** (true) if the operator class is the default for **opcintype**. Otherwise, the value is **f** (false).
opckeytype	oid	OID in **PG_TYPE**	Type of data stored in an index, or zero if same as **opcintype**

An operator class's **opcmethod** must match the **opfmethod** of its containing operator family.

12.2.14.49 PG_OPERATOR

PG_OPERATOR records information about operators.

Table 12-97 PG_OPERATOR columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified).
oprname	name	-	Name of an operator.
oprnamespace	oid	**oid** in **PG_NAMESPACE**	OID of the namespace that contains the operator.
oprowner	oid	OID in **PG_AUTHID**	Owner of the operator.
oprkind	"char"	-	• **b**: infix (both sides) • **l**: prefix (left side) • **r**: suffix (right side)
oprcanmerge	Boolean	-	Specifies whether the operator supports merge joins. • **t (true)**: yes • **f (false)**: no
oprcanhash	Boolean	-	Specifies whether the operator supports hash joins. • **t (true)**: yes • **f (false)**: no
oprleft	oid	OID in **PG_TYPE**	Type of the left operand.
oprright	oid	OID in **PG_TYPE**	Type of the right operand.

Name	Type	Reference	Description
oprresult	oid	OID in **PG_TYPE**	Type of the result.
oprcom	oid	OID in **PG_OPERATOR**	Exchange character of this operator if any; otherwise, **0**.
oprnegate	oid	OID in **PG_OPERATOR**	Negator of this operator if any; otherwise, **0**.
oprcode	regproc	**proname** in **PG_PROC**	Function that implements the operator
oprrest	regproc	**proname** in **PG_PROC**	Restriction selectivity estimation function for the operator.
oprjoin	regproc	**proname** in **PG_PROC**	Join selectivity estimation function for the operator.

12.2.14.50 PG_OPFAMILY

PG_OPFAMILY defines operator families.

Each operator family is a collection of operators and associated support routines that implement semantics specified for a particular index access method. Furthermore, the operators in a family are all compatible, in a way that is specified by the access method. The operator family allows cross-data-type operators to be used with indexes and to be reasoned about using knowledge of access method semantics.

Table 12-98 PG_OPFAMILY columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
opfmethod	oid	**PG_AM**.oid	Index access method used by an operator family
opfname	name	-	Name of the operator family
opfnamespace	oid	**PG_NAMESPACE**.oid	Namespace of the operator family
opfowner	oid	**PG_AUTHID**.oid	Owner of the operator family

The majority of the information defining an operator family is not in its **PG_OPFAMILY** row, but in the associated rows in **PG_AMOP**, **PG_AMPROC**, and **PG_OPCLASS**.

12.2.14.51 PG_PLTEMPLATE

PG_PLTEMPLATE records template information for procedural languages.

Table 12-99 PG_PLTEMPLATE columns

Name	Type	Description
tmplname	name	Name of the language for which this template is used.
tmpltrusted	Boolean	The value is **true** if the language is considered trusted. Otherwise, the value is **false**.
tmpldbacreate	Boolean	The value is **true** if the language is created by the owner of the database. Otherwise, the value is **false**.
tmplhandler	text	Name of the call handler function.
tmplinline	text	Name of the anonymous block handler (**NULL** if no name of the block handler exists).
tmplvalidator	text	Name of the verification function (**NULL** if no verification function is available).
tmpllibrary	text	Path of the shared library that implements languages.
tmplacl	aclitem[]	Access permissions for template (not yet used).

12.2.14.52 PG_PROC

PG_PROC stores information about functions or stored procedures.

Table 12-100 PG_PROC columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
proname	name	Function name.
pronamespace	oid	OID of the namespace that contains the function.

Name	Type	Description
proowner	oid	Owner of the function.
prolang	oid	Implementation language or call interface of the function.
procost	real	Estimated execution cost.
prorows	real	Estimated number of rows that are influenced.
provariadic	oid	Data type of parameter element.
protransform	regproc	Simplified call method for the function.
proisagg	Boolean	Specifies whether the function is an aggregate function. ● **t** (true): yes ● **f** (false): no
proiswindow	Boolean	Specifies whether the function is a window function. ● **t** (true): yes ● **f** (false): no
prosecdef	Boolean	Specifies whether the function is a security definer (or a setuid function). ● **t** (true): yes ● **f** (false): no
proleakproof	Boolean	Specifies whether the function has side effects. If no leakproof treatment is provided for parameters, the function throws errors. ● **t** (true): There is no side effect. ● **f** (false): There are side effects.
proisstrict	Boolean	● **t** (true): When this function is used, the function does not invoke the function and returns null if the input parameter is null. ● **f** (false): When this function is used, it is invoked even if the input parameter is null. Therefore, this function must process null input.
proretset	Boolean	Specifies whether the return value of the function is a set (that is, multiple values of the specified data type). ● **true**: The return value of the function is a set. ● **false**: The return value of the function is not a set.

Name	Type	Description
provolatile	"char"	Specifies whether the function's result depends only on its input parameters, or is affected by outside factors. ● **i**: immutable functions, which always deliver the same result for the same inputs. ● **s**: stable functions, whose results (for fixed inputs) do not change within a scan. ● **v**: volatile functions, whose results may change at any time. Use **v** also for functions with side-effects, so that the engine cannot get optimized if volatile functions are called.
pronargs	smallint	Number of parameters.
pronargdefaults	smallint	Number of parameters that have default values.
prorettype	oid	Data type of return values.
proargtypes	oidvector	Array that stores the data types of function parameters. This array includes only input parameters (including **INOUT** parameters), and indicates the call signature (interface) of the function.
proallargtypes	oid[]	Array that contains the data types of function parameters. This array includes all parameter types (including **OUT** and **INOUT** parameters); however, if all the parameters are **IN** parameters, this column is null. Note that array subscripting is 1-based, whereas for historical reasons, **proargtypes** is subscripted from 0.
proargmodes	"char"[]	Array with the modes of the function parameters, encoded as follows: ● **i** indicates the **IN** parameter. ● **o** indicates the **OUT** parameter. ● **b** indicates the **INOUT** parameter. ● **v** indicates the **VARIADIC** parameter. If all the parameters are **IN** parameters, this column is null. Note that subscripts correspond to positions of **proallargtypes**, not **proargtypes**.
proargnames	text[]	Array that stores the names of the function parameters. Parameters without a name are set to empty strings in the array. If none of the parameters have a name, this column is null. Note that subscripts correspond to positions of **proallargtypes**, not **proargtypes**.

Name	Type	Description
proargdefaults	pg_node_tree	Expression tree of the default value. This is the list of **pronargdefaults** elements.
prosrc	text	A definition that describes a function or stored procedure. In an interpreting language, it is the function source code, a connection symbol, a file name, or any body content specified when a function or stored procedure is created, depending on how a language or call is used.
probin	text	Additional information about how to call the function. Again, the interpretation is language-specific.
proconfig	text[]	Function's local settings for run-time configuration variables.
proacl	aclitem[]	Access permissions. For details, see **GRANT** and **REVOKE**.
prodefaultargpos	int2vector	Position of the input parameter of a function with a default value.
fencedmode	Boolean	Execution mode of a function, indicating whether the function is executed in fence or not fence mode. ● **true**: fence mode. In this mode, the function is executed in the fork process. ● **false**: not fence mode. For built-in functions in the system, **fencedmode** is set to **false**, indicating the not fence mode.
proshippable	Boolean	Specifies whether the function can be pushed down to database nodes. The default value is **false**. ● Functions of the IMMUTABLE type can always be pushed down to the database nodes. ● A STABLE or VOLATILE function can be pushed down to the database nodes only if SHIPPABLE is specified for it.
propackage	Boolean	Specifies whether the function supports overloading. The default value is **false**. ● **t** (true): supported. ● **f** (false): not supported.

Name	Type	Description
prokind	"char"	Specifies whether the object is a function or a stored procedure. ● **'f'**: The object is a function. ● **'p'**: The object is a stored procedure.
proargsrc	text	Describes the parameter input strings of functions or stored procedures that are A-compatible syntax, including parameter comments. The default value is **NULL**.
proisprivate	Boolean	Specifies whether a function is a private function in the package. The default value is **false**.
propackageid	oid	OID of the package to which the function belongs. If the function is not in the package, the value is **0**.
proargtypesext	oidvector_extend	Data type array used to store function parameters when there are a large number of function parameters. This array includes only input parameters (including **INOUT** parameters), and indicates the call signature (interface) of the function.
prodefaultargposext	int2vector_extend	Position of the input parameter with a default value when the function has a large number of parameters.
allargtypes	oidvector	Array for storing the data types of stored procedure parameters, including all parameters (input parameters, output parameters, and **INOUT** parameters) of the stored procedure.
allargtypesext	oidvector_extend	Array for storing the data types of stored procedure parameters when the number of function parameters is greater than 666. The array contains all parameters (including input parameters, output parameters, and **INOUT** parameters).

NOTE

When a function is created, data is inserted into the PG_PROC catalog to update the index. When there are a large number of input and output parameters, the index length may exceed one third of the page length. As a result, the error "Index row size xxx exceeds maximum xxx for index 'pg_proc_proname_all_args_nsp_index'" may be reported as expected. You can reduce the number of parameters to avoid this error.

12.2.14.53 PG_RANGE

PG_RANGE records information about range types, except for records of types in **PG_TYPE**.

Table 12-101 PG_RANGE columns

Name	Type	Reference	Description
rngtypid	oid	OID in **PG_TYPE**	OID of the range type.
rngsubtype	oid	OID in **PG_TYPE**	OID of the element type (subtype) of this range type.
rngcollation	oid	OID in **PG_COLLATION**	OID of the collation used for range comparisons (**0** if none).
rngsubopc	oid	OID in **PG_OPCLASS**	OID of the subtype's operator class used for range comparisons.
rngcanonical	regproc	**proname** in **PG_PROC**	Name of the function to convert a range value into canonical form (**0** if none).
rngsubdiff	regproc	**proname** in **PG_PROC**	Name of the function used to calculate the difference between two elements in a range. The return value of the function is of the double-precision type. If the function does not exist, the value of **rngsubdiff** is **0**.

If the element type is discrete, **rngcanonical** determines the collation used for the range type. If the element type is not collatable, **rngsubopc** determines the collation used for the range type. If the element type is collatable, **rngsubopc** and **rngcollation** determine the collation used for the range type.

12.2.14.54 PG_REPLICATION_ORIGIN

PG_REPLICATION_ORIGIN contains all created replication sources. This catalog is shared among all databases of a database instance. That is, there is only one copy for each instance, not for each database.

Table 12-102 PG_REPLICATION_ORIGIN columns

Name	Type	Description
roident	oid	Unique replication source identifier within a cluster.

Name	Type	Description
roname	text	External user-defined replication source name.

12.2.14.55 PG_RESOURCE_POOL

PG_RESOURCE_POOL provides information about database resource pools.

Table 12-103 PG_RESOURCE_POOL columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
respool_name	name	Name of the resource pool.
mem_percent	integer	Percentage of the memory configuration
cpu_affinity	bigint	Value of cores bound to the CPU
control_group	name	Name of the Cgroup where the resource pool is located
active_statements	integer	Maximum number of concurrent statements in the resource pool
max_dop	integer	Maximum scanning concurrency during data redistribution. This column is used only for scaling.
memory_limit	name	Maximum memory of the resource pool
parentid	oid	OID of the parent resource pool
io_limits	integer	Upper limit of IOPS. It is counted by ones and by 10 thousands.
io_priority	name	Specifies the I/O priority set for jobs that consume many I/O resources. It takes effect when the I/O usage reaches 90%.
nodegroup	name	Name of the logical database to which the resource pool belongs This column is not supported in the centralized system.

Name	Type	Description
is_foreign	Boolean	Specifies whether the resource pool can be used for users outside the logical database. This column is not supported in the centralized system. • **true**: The resource pool controls the resources of common users who do not belong to the current resource pool. • **false**: The resources of common users who do not belong to the current resource pool are not controlled.
max_worker	integer	Concurrency in a table during data redistribution. This column is used only for scaling.
max_connections	integer	Maximum number of connections that can be used by a resource pool.
max_dynamic_memory	name	Maximum dynamic memory that can be used by a resource pool.
max_shared_memory	name	Maximum shared memory that can be used by a resource pool.
max_concurrency	integer	Maximum number of concurrent requests that can be used by a resource pool.

Note: **max_dop** and **max_worker** are used for scaling and are not applicable to the centralized deployment.

12.2.14.56 PG_REWRITE

PG_REWRITE records rewrite rules defined for tables and views.

Table 12-104 PG_REWRITE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
rulename	name	Rule name.
ev_class	oid	Name of the table that uses the rule.
ev_attr	smallint	Column to which this rule applies (always **0** to indicate the entire table).

Name	Type	Description
ev_type	"char"	Event type for the rule. • **1**: SELECT • **2**: UPDATE • **3**: INSERT • **4**: DELETE
ev_enabled	"char"	Controls the mode in which the rule is triggered. • **O**: The rule is triggered in origin and local modes. • **D**: The rule is disabled. • **R**: The rule is triggered in replica mode. • **A**: The rule is always triggered.
is_instead	Boolean	The value is **true** if the rule is of the **INSTEAD** type. Otherwise, the value is **false**.
ev_qual	pg_node_tree	Expression tree (in the form of a nodeToString() representation) for the rule's qualifying condition.
ev_action	pg_node_tree	Query tree (in the form of a nodeToString() representation) for the rule's action.

12.2.14.57 PG_SHDEPEND

PG_SHDEPEND records the dependency between database objects and shared objects, such as roles. Based on this information, GaussDB can ensure that those objects are unreferenced before attempting to delete them.

See also **PG_DEPEND**, which provides a similar function for dependencies involving objects within a single database.

PG_SHDEPEND is shared among all databases of a database instance. That is, there is only one PG_SHDEPEND for each instance, not for each database.

Table 12-105 PG_SHDEPEND columns

Name	Type	Reference	Description
dbid	oid	OID in **PG_DATABASE**	OID of the database where a dependent object is (**0** for a shared object)
classid	oid	OID in **PG_CLASS**	OID of the system catalog where a dependent object resides
objid	oid	Any OID column	OID of the dependent object

Name	Type	Reference	Description
objsubid	integer	-	Column number of a table column (the table to which the column points can be determined based on **objid** and **classid**). The value is **0** for all other object types.
refclassid	oid	OID in PG_CLASS	OID of the system catalog (must be a shared catalog) where a referenced object is located.
refobjid	oid	Any OID column	OID of the referenced object.
deptype	"char"	-	Code segment defining the specific semantics of this dependency relationship. The value can be **o**, **a**, **p**, **d**, or **l**. For details, see the following text.
objfile	text	-	Path of a user-defined function library file

In all cases, a PG_SHDEPEND entry indicates that the referenced object cannot be dropped without also dropping the dependent object. However, there are several subflavors identified by **deptype**:

- SHARED_DEPENDENCY_OWNER (o)

  The referenced object (which must be a role) is the owner of the dependent object.

- SHARED_DEPENDENCY_ACL (a)

  The referenced object (which must be a role) is mentioned in the access control list (ACL) of the dependent object. SHARED_DEPENDENCY_ACL does not add a record to the owner of the object because the owner will have a SHARED_DEPENDENCY_OWNER record.

- SHARED_DEPENDENCY_PIN (p)

  This type of record indicates that the system itself depends on the depended object. Therefore, such an object cannot be deleted. Entries of this type are created only by **initdb**. The columns for the dependent object contain zeroes.

- SHARED_DEPENDENCY_ DBPRIV(d)

  The referenced object (must be a role) has the ANY permission on the dependent object (the specified OID of the dependent object corresponds to a row in the 12.2.10.1 GS_DB_PRIVILEGE system catalog).

- SHARED_DEPENDENCY_ SECLABEL(l)

  The referenced object (must be a security label) is applied to the dependent object.

12.2.14.58 PG_SHDESCRIPTION

PG_SHDESCRIPTION records optional comments for shared database objects. Descriptions can be manipulated with the **COMMENT** command and viewed with the **\d** command.

See also PG_DESCRIPTION, which provides a similar function for descriptions involving objects within a single database.

PG_SHDESCRIPTION is shared among all databases of a database instance. That is, there is only one PG_SHDESCRIPTION for each instance, not for each database.

Table 12-106 PG_SHDESCRIPTION columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the described object.
classoid	oid	OID in PG_CLASS	OID of the system catalog where the object appears.
description	text	-	Description of the object.

12.2.14.59 PG_SET

PG_SET records metadata defined by the SET data type.

Table 12-107 PG_SET columns

Name	Type	Description
settypid	oid	OID of the SET data type
setnum	tinyint	Number of members of the SET data type. A maximum of 64 members are supported.
setsortorder	tinyint	Sorting order of members when the SET data type is defined. The value starts from 0.
setlabel	text	Member name of the SET data type

12.2.14.60 PG_STATISTIC

PG_STATISTIC stores statistics about tables and index columns in a database. By default, only the system administrator can access the system catalog. Common users can access the system catalog only after being authorized.

Table 12-108 PG_STATISTIC columns

Name	Type	Description
starelid	oid	Table or index that the described column belongs to.

Name	Type	Description
starelkind	"char"	Type of an object.
staattnum	smallint	Number of the described column in the table, starting from 1.
stainherit	Boolean	Specifies whether to collect statistics for objects that have inheritance relationship.
stanullfrac	real	Percentage of column entries that are null.
stawidth	integer	Average stored width, in bytes, of non-null entries.
stadistinct	real	Number of distinct, non-null data values in the column for database nodes. ● A value greater than 0 is the actual number of distinct values. ● A value less than 0 is the ratio of the **distinct** value to the total number of rows. For example, if **stadistinct** is **-0.5**, the actual **distinct** value is the total number of rows multiplied by 0.5. ● The value **0** indicates that the number of distinct values is unknown.
stakindN	smallint	Code number stating that the type of statistics is stored in slot *N* of the *pg_statistic* row. The value of *N* ranges from 1 to 5.
staopN	oid	Operator used to generate the statistics stored in slot *N*. For example, a histogram slot shows the < operator that defines the sort order of the data. The value of *N* ranges from 1 to 5.
stanumbersN	real[]	Numerical statistics of the appropriate type for slot *N*. The value is **NULL** if the slot does not involve numerical values. The value of *N* ranges from 1 to 5.
stavaluesN	anyarray	Column data values of the appropriate type for slot *N*. The value is **NULL** if the slot type does not store any data values. Each array's element values are actually of the specific column's data type, so there is no way to define these columns' type more specifically than anyarray. The value of *N* ranges from 1 to 5.

Name	Type	Description
stadndistinct	real	Number of unique non-null data values in the **DN1** column. • A value greater than 0 is the actual number of distinct values. • A value less than 0 is the ratio of the **distinct** value to the total number of rows. For example, if **stadistinct** is **-0.5**, the actual **distinct** value is the total number of rows multiplied by 0.5. • The value **0** indicates that the number of distinct values is unknown.
staextinfo	text	Information about extension statistics. This is reserved.
stastate	"char"	Specifies whether the statistics are locked. If the statistics are locked, they cannot be updated. • **l**: locked • **u**: unlocked

NOTICE

PG_STATISTIC stores sensitive information about statistical objects, such as MCVs. The system administrator and authorized users can access the PG_STATISTIC system catalog to query the sensitive information about the statistical objects.

12.2.14.61 PG_STATISTIC_EXT

PG_STATISTIC_EXT displays extended statistics of tables in a database, such as statistics of multiple columns. Statistics of expressions will be supported later. You can specify the extended statistics to be collected. This system catalog is accessible only to the system administrator.

Table 12-109 PG_STATISTIC_EXT columns

Name	Type	Description
starelid	oid	Table or index that the described column belongs to.
starelkind	char	Type of the object to which a table belongs. **'c'** indicates an ordinary table, and **'p'** indicates a partitioned table.
stainherit	Boolean	Specifies whether to collect statistics for objects that have inheritance relationship.
stanullfrac	real	Percentage of column entries that are null.
stawidth	integer	Average stored width, in bytes, of non-null entries.

Name	Type	Description
stadistinct	real	Number of distinct, non-null data values in the column for database nodes. • A value greater than 0 is the actual number of distinct values. • A value less than 0 is the ratio of the **distinct** value to the total number of rows. For example, if **stadistinct** is **-0.5**, the actual **distinct** value is the total number of rows multiplied by 0.5. • The value **0** indicates that the number of distinct values is unknown.
stadndistinct	real	Number of unique non-null data values in the **DN1** column. • A value greater than 0 is the actual number of distinct values. • A value less than 0 is the ratio of the **distinct** value to the total number of rows. For example, if **stadistinct** is **-0.5**, the actual **distinct** value is the total number of rows multiplied by 0.5. • The value **0** indicates that the number of distinct values is unknown.
stakindN	smallint	Code number stating that the type of statistics is stored in slot *N* of the *pg_statistic* row. The value of *N* ranges from 1 to 5.
staopN	oid	Operator used to generate the statistics stored in slot *N*. For example, a histogram slot shows the < operator that defines the sort order of the data. The value of *N* ranges from 1 to 5.
stakey	int2vector	Array of a column ID.
stanumbersN	real[]	Numerical statistics of the appropriate type for slot *N*. The value is **NULL** if the slot does not involve numerical values. The value of *N* ranges from 1 to 5.
stavaluesN	anyarray	Column data values of the appropriate type for slot *N*. The value is **NULL** if the slot type does not store any data values. Each array's element values are actually of the specific column's data type so there is no way to define these columns' type more specifically than anyarray. The value of *N* ranges from 1 to 5.
staexprs	pg_node_tree	Expression corresponding to the extended statistics information.

Name	Type	Description
stasource	char	Source of extended statistics: • **'a'**: indicates that the statistics data is automatically created, which is controlled by the GUC parameter **auto_statistic_ext_columns**. • **'m'**: indicates that a user manually creates the statistics data using analyze tablename ((column list)) or alter table tablename add statistics ((column list)).
stastatus	char	Status of extended statistics: • **'a'**: active and available. • **'d'**: disabled. Related information is not collected, and the optimizer does not use the data when generating a plan. You can use the alter table tablename disable/enable statistics((column list)) syntax to modify the status of extended statistics.
staextname	name	Alias of the multi-column group of multi-column statistics.
stastate	"char"	Specifies whether the statistics are locked. If the statistics are locked, they cannot be updated. • **l**: locked • **u**: unlocked

NOTICE

PG_STATISTIC_EXT stores sensitive information about statistical objects, such as MCVs. The system administrator and authorized users can access the PG_STATISTIC_EXT system catalog to query the sensitive information about the statistical objects.

12.2.14.62 PG_SYNONYM

PG_SYNONYM records the mapping between synonym object names and other database object names.

Table 12-110 PG_SYNONYM columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
synname	name	Synonym name.

Name	Type	Description
synnamespace	oid	OID of the namespace that contains a synonym. The OID of the PUBLIC synonym namespace is 0.
synowner	oid	Owner of a synonym, usually the OID of the user who created it The owner of the PUBLIC synonym is 0.
synobjschema	name	Schema name specified by an associated object.
synobjname	name	Name of an associated object.
syndblinkname	name	Name of the associated DATABASE LINK object.

12.2.14.63 PG_TABLESPACE

PG_TABLESPACE records tablespace information.

Table 12-111 PG_TABLESPACE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
spcname	name	Tablespace name.
spcowner	oid	Owner of the tablespace, usually the user who created it.
spcacl	aclitem[]	Access permissions. For details, see **GRANT** and **REVOKE**.
spcoptions	text[]	Options of the tablespace.
spcmaxsize	text	Maximum size of the available disk space, in bytes.
relative	Boolean	Specifies whether the storage path specified by the tablespace is a relative path. • **t** (true): yes • **f** (false): no

12.2.14.64 PG_TRIGGER

PG_TRIGGER records trigger information.

Table 12-112 PG_TRIGGER columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
tgrelid	oid	OID of the table where the trigger is located.
tgname	name	Trigger name.
tgfoid	oid	Function to be invoked by the trigger.
tgtype	smallint	Trigger type.
tgenabled	"char"	• **O**: The trigger is triggered in origin and local modes. • **D**: The trigger is disabled. • **R**: The trigger is triggered in replica mode. • **A**: The trigger is always triggered.
tgisinternal	Boolean	Internal trigger ID. If the value is **true**, it indicates an internal trigger.
tgconstrrelid	oid	Table referenced by the integrity constraint.
tgconstrindid	oid	Index of the integrity constraint.
tgconstraint	oid	OID of the constraint trigger in PG_CONSTRAINT.
tgdeferrable	Boolean	Specifies whether the constraint trigger is of the DEFERRABLE type. • **t** (true): yes • **f** (false): no
tginitdeferred	Boolean	Specifies whether the trigger is of the INITIALLY DEFERRED type. • **t** (true): yes • **f** (false): no
tgnargs	smallint	Number of input parameters of the trigger function.
tgattr	int2vector	Column ID specified by the trigger. If no column is specified, an empty array is used.
tgargs	bytea	Parameter transferred to the trigger.
tgqual	pg_node_tree	WHEN condition of the trigger (**NULL** if the WHEN condition does not exist).
tgowner	oid	Trigger owner.

12.2.14.65 PG_TS_CONFIG

PG_TS_CONFIG contains entries representing text search configurations. Each of these configurations contains a list of specific text search parsers and dictionary mappings.

The parser is shown in the PG_TS_CONFIG entry, but the token-to-dictionary mapping is defined by subsidiary entries in **PG_TS_CONFIG_MAP**.

Table 12-113 PG_TS_CONFIG columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified).
cfgname	name	-	Text search configuration name.
cfgnamespace	oid	**oid** in **PG_NAMESPACE**	OID of the namespace that contains the configuration.
cfgowner	oid	**PG_AUTHID**.oid	Owner of the configuration.
cfgparser	oid	**PG_TS_PARSER**.oid	OID of the text search parser for this configuration.
cfoptions	text[]	-	Configuration options.

12.2.14.66 PG_TS_CONFIG_MAP

PG_TS_CONFIG_MAP contains parser symbol mappings for each text search configuration.

Table 12-114 PG_TS_CONFIG_MAP columns

Name	Type	Reference	Description
mapcfg	oid	OID in **PG_TS_CONFIG**	OID of the **PG_TS_CONFIG** entry owning this map entry.
maptokentype	integer	-	Token type generated by the configuration's parser.
mapseqno	integer	-	Sequence number of a token type when the values of **mapcfg** or **maptokentype** are the same.
mapdict	oid	OID in **PG_TS_DICT**	OID of the text search dictionary to consult.

12.2.14.67 PG_TS_DICT

PG_TS_DICT contains entries that define text search dictionaries. A dictionary depends on a text search template, which specifies all the implementation functions needed; the dictionary itself provides values for the user-settable parameters supported by the template.

This division of labor allows dictionaries to be created by unprivileged users. The parameters are specified by a text string **dictinitoption**, whose format and meaning vary depending on the template.

Table 12-115 PG_TS_DICT columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified)
dictname	name	-	Text search dictionary name
dictnamespace	oid	PG_NAMESPACE.oid	OID of the namespace that contains the dictionary
dictowner	oid	PG_AUTHID.oid	Owner of the dictionary
dicttemplate	oid	PG_TS_TEMPLATE.oid	OID of the text search template for the dictionary
dictinitoption	text	-	Initialization option string for the template

12.2.14.68 PG_TS_PARSER

PG_TS_PARSER contains definition of text search parsers. A parser is responsible for splitting input text into lexemes and assigning a token type to each lexeme. Because the parser must be implemented through functions at the C language level, the new parser must be created by the database system administrator.

Table 12-116 PG_TS_PARSER columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified).
prsname	name	-	Text search parser name.
prsnamespace	oid	**oid** in PG_NAMESPACE	OID of the namespace that contains the parser.

Name	Type	Reference	Description
prsstart	regproc	**proname** in PG_PROC	Name of the parser's startup function.
prstoken	regproc	**proname** in PG_PROC	Name of the parser's next-token function.
prsend	regproc	**proname** in PG_PROC	Name of the parser's shutdown function.
prsheadline	regproc	**proname** in PG_PROC	Name of the parser's headline function.
prslextype	regproc	**proname** in PG_PROC	Name of the parser's lextype function.

12.2.14.69 PG_TS_TEMPLATE

PG_TS_TEMPLATE contains entries defining text search templates. A template provides a framework for text search dictionaries. Since a template must be implemented by C-language-level functions, templates can be created only by database administrators.

Table 12-117 PG_TS_TEMPLATE columns

Name	Type	Reference	Description
oid	oid	-	Row identifier (hidden attribute, which must be specified).
tmplname	name	-	Text search template name
tmplnamespace	oid	**oid** in PG_NAMESPACE	OID of the namespace that contains the template
tmplinit	regproc	**proname** in PG_PROC	Name of the template's initialization function
tmpllexize	regproc	**proname** in PG_PROC	Name of the template's lexize function

12.2.14.70 PG_TYPE

PG_TYPE stores information about data types.

Table 12-118 PG_TYPE columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).
typname	name	Data type name.
typnamespace	oid	OID of the namespace that contains the type.
typowner	oid	Owner of the type.
typlen	smallint	Number of bytes in the internal representation of the type for a fixed-size type. It is a negative number for a variable-length type. • The value **–1** indicates a "varlena" type (one that has a length word). • The value **–2** indicates a null-terminated C string.
typbyval	Boolean	• **true**: A value of this type is transferred by value internally. • **false**: A value of this type is transferred by reference internally. If **typlen** of this type is not **1**, **2**, **4**, or **8**, you are advised to transfer a reference value for **typbyval**. You can also transfer a value for **typbyval**. Variable-length types are usually passed by reference or by value.
typtype	"char"	• **b**: basic type. • **c**: composite type (for example, the row type of a table). • **d**: domain type. • **p**: pseudo type. • **u**: undefined type. • **o**: set type. For details, see **typrelid** and **typbasetype**.
typcategory	"char"	Fuzzy classification of data types, which can be used by parsers as the basis for data conversion.
typispreferred	Boolean	• **true**: Data is converted when it complies with the conversion rule specified by **typcategory**. • **false**: Data is not converted.
typisdefined	Boolean	• **true**: The type has been defined. • **false**: The placeholder of an undefined type is used. In this case, there is no reliable information except the name, namespace, and OID of the type.

Name	Type	Description
typdelim	"char"	Character that separates two values of this type when parsing an array input. Note that the delimiter is associated with the array element data type, not the array data type.
typrelid	oid	If this is a composite type (see **typtype**), then this column points to the **PG_CLASS** entry that defines the corresponding table. For a free-standing composite type, the PG_CLASS entry does not represent a table, but it is required for the type's **PG_ATTRIBUTE** entries to link to. It is **0** for non-composite type.
typelem	oid	If **typelem** is not **0**, it identifies another row in PG_TYPE. The current type can be described as an array yielding values of type **typelem**. A "true" array type has a variable length (**typlen** = **–1**), but some fixed-length types (**typlen** > **0**) also have non-zero **typelem**, for example **name** and **point**. If a fixed-length type has a **typelem**, its internal representation must be a number of values of the **typelem** data type with no other data. Variable-length array types have a header defined by the array subroutines.
typarray	oid	If the value is not **0**, the corresponding type record is available in PG_TYPE.
typinput	regproc	Input conversion function (text format)
typoutput	regproc	Output conversion function (text format)
typreceive	regproc	Input conversion function (binary format); **0** for non-input conversion function
typsend	regproc	output conversion function (binary format); **0** for non-output conversion function
typmodin	regproc	Input type modifier function; **0** if none.
typmodout	regproc	Output type modifier function; **0** if none.
typanalyze	regproc	Custom ANALYZE function; **0** if the standard function is used.

Name	Type	Description
typalign	"char"	Alignment required when storing a value of this type. It applies to storage on disks as well as most representations of the value. When multiple values are stored consecutively, such as in the representation of a complete row on disk, padding is inserted before a data of this type so that it begins on the specified boundary. The alignment reference is the beginning of the first datum in the sequence. Possible values are: • **c**: char alignment, that is, no alignment required • **s**: short alignment (2 bytes on most machines) • **i**: integer alignment (4 bytes on most machines) • **d**: double alignment (8 bytes on many machines, but by no means all) **NOTICE** For types used in system tables, the size and alignment defined in PG_TYPE must agree with the way that the compiler lays out the column in a structure representing a table row.
typstorage	"char"	**typstorage** tells for varlena types (those with **typlen = –1**) if the type is prepared for toasting and what the default strategy for attributes of this type should be. Possible values are: • **p**: Values are always stored plain. • **e**: Values can be stored in a secondary relationship (if the relation has one, see **reltoastrelid** in 12.2.15.28 PG_CLASS). • **m**: Values can be stored compressed inline. • **x**: Values can be stored compressed inline or stored in secondary storage. **NOTICE** **m** domains can also be moved out to secondary storage, but only as a last resort (**e** and **x** domains are moved first).
typnotnull	Boolean	Specifies whether the type has a NOTNULL constraint. Currently, it is used for domains only.
typbasetype	oid	If this is a domain (see **typtype**), then **typbasetype** identifies the type that this one is based on. The value is **0** if this type is not a derived type.

Name	Type	Description
typtypmod	integer	Records the **typtypmod** to be applied to domains' base types by domains (the value is **–1** if the base type does not use **typmod**). This is **–1** if this type is not a domain. If the type is array, **typtypmod** indicates the maximum capacity of the array type. If the type is index-by table collection, **typtypmod** indicates the maximum index length of the collection type.
typndims	integer	If a field is an array, **typndims** is the number of array dimensions. This is **0** for types other than domains over array types.
typcollation	oid	Sequence rule for specified types. For details about the value, see the system catalog in PG_COLLATION. (**0** if sequencing is not supported)
typdefaultbin	pg_node_tree	**nodeToString()** representation of a default expression for the type if the value is non-null. Currently, this column is only used for domains.
typdefault	text	The value is **NULL** if a type has no associated default value. • If **typdefaultbin** is not **NULL**, **typdefault** must contain a default expression represented by **typdefaultbin**. • If **typdefaultbin** is **NULL** and **typdefault** is not, then **typdefault** is the external representation of the type's default value, which can be fed to the type's input converter to produce a constant.
typacl	aclitem[]	Access permission
typelemmod	integer	• **-1**: **typmod** is not required for the element type corresponding to the set and array types. • Value greater than or equal to 0: **typmod** of the element type corresponding to the set and array types is used. • **NULL**: **typmod** of the element type corresponding to the set and array types is unknown. **typmod** usually refers to the maximum length of a type.

12.2.14.71 PGXC_CLASS

PGXC_CLASS records replicated or distributed information for each table. PGXC_CLASS can only be used to query table definitions in centralized mode.

Table 12-119 PGXC_CLASS columns

Name	Type	Description
pcrelid	oid	Table OID
pclocatortype	"char"	Locator type • **H**: Hash • **G**: Range • **L**: List • **M**: Modulo • **N**: Round Robin • **R**: Replication
pchashalgorithm	smallint	Distributed tuple using the hash algorithm • **1**: default hash algorithm. • **2**: MURMURHASH algorithm.
pchashbuckets	smallint	Value of a hash container
pgroup	name	Name of the node
redistributed	"char"	Indicates that a table has been redistributed.
redis_order	integer	Redistribution sequence. Tables whose values are **0** will not be redistributed in this round of redistribution.
pcattnum	int2vector	Column number used as a distributed key
nodeoids	oidvector_extend	List of distributed table node OIDs
options	text	Extension status information. This is a reserved column in the system.
diskey	text	Not supported. Its value is **NULL**.
diskeyexprs	pg_node_tree	Not supported. Its value is **NULL**.

12.2.14.72 PGXC_GROUP

PGXC_GROUP records information about node groups. PGXC_GROUP can only be used to query table definitions in the centralized system.

Table 12-120 PGXC_GROUP columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute, which must be specified).

Name	Type	Description
group_name	name	Node group name.
in_redistribution	"char"	Specifies whether redistribution is required. Valid value: • **n**: The node group is not redistributed. • **y**: The source node group is in redistribution. • **t**: The destination node group is in redistribution.
group_members	oidvector_extend	Node OID list of the node group.
group_buckets	text	Distributed data bucket group.
is_installation	Boolean	Specifies whether to install a sub-database instance. • **t** (true): yes • **f** (false): no
group_acl	aclitem[]	Access permission
group_kind	"char"	Node group type. The options are as follows: • **i**: installation node group • **n**: node group in a common non-logical database instance • **v**: node group in a logical database instance • **e**: elastic database instance
group_parent	oid	For a child node group, this field indicates the OID of the parent node group. For a parent node group, this field is left blank.
bucket_map	text	Not supported. The value is **NULL**.

12.2.14.73 PGXC_SLICE

PGXC_SLICE is a system catalog created for recording range distribution and list distribution details. Currently, range interval cannot be used to automatically scale out shards. It is reserved in the system catalog. In the centralized system, only the definition of this table can be queried.

Table 12-121 PGXC_SLICE columns

Name	Type	Description
relname	name	Table name or shard name, which is distinguished by **type**.

Name	Type	Description
type	"char"	• **'t'**: **relname** is the table name. • **'s'**: **relname** is the shard name.
strategy	"char"	• **'r'**: range distribution table. • **'l'**: list distribution table. This value will be extended for subsequent interval shards.
relid	oid	OID of the distribution table to which the tuple belongs.
referenceoid	oid	OID of the referenced distribution table, which is used for slice reference table creation syntax.
sindex	integer	Position of the current boundary in a shard when the table is a list distribution table.
interval	text[]	Reserved column.
transitboundary	text[]	Reserved column.
transitno	integer	Reserved column.
nodeoid	oid	When **relname** is set to a shard name, **nodeoid** indicates the OID of the DN where the shard data is stored.
boundaries	text[]	When **relname** is set to a shard name, this parameter indicates the boundary value of the shard.
specified	Boolean	Specifies whether the DN corresponding to the current shard is explicitly specified in the DDL.
sliceorder	integer	User-defined shard sequence.

12.2.14.74 PLAN_TABLE_DATA

PLAN_TABLE_DATA stores plan information collected by EXPLAIN PLAN. Different from the PLAN_TABLE view, the system catalog PLAN_TABLE_DATA stores EXPLAIN PLAN information collected by all sessions and users.

Table 12-122 PLAN_TABLE_DATA columns

Name	Type	Description
session_id	text	Session that inserts the data. Its value consists of a service thread start timestamp and a service thread ID. Values are constrained by **NOT NULL**.

Name	Type	Description
user_id	oid	User who inserts the data. Values are constrained by **NOT NULL**.
statement_id	character varying(30)	Query tag specified by a user.
plan_id	bigint	Query ID The ID is automatically generated in the plan generation phase and is used by kernel engineers for debugging.
id	integer	Node ID in a plan.
operation	character varying(30)	Operation description.
options	character varying(255)	Operation action.
object_name	name	Name of an operated object. It is defined by users.
object_type	character varying(30)	Object type.
object_owner	name	Schema to which the object belongs. It is defined by users.
projection	character varying(4000)	Returned column information.
cost	double precision	Execution cost estimated by the optimizer for an operator.
cardinality	double precision	Number of rows estimated by the optimizer for an operator.

NOTE

- PLAN_TABLE_DATA records data of all users and sessions on the current node. Only administrators can access all the data. Common users can view their own data in the **PLAN_TABLE** view.
- Data is automatically inserted into PLAN_TABLE_DATA after **EXPLAIN PLAN** is executed. Therefore, do not manually insert data into or update data in PLAN_TABLE_DATA. Otherwise, data in PLAN_TABLE_DATA may be disordered. To delete data from PLAN_TABLE_DATA, you are advised to use the **PLAN_TABLE** view.
- Information in the **statement_id**, **object_name**, **object_owner**, and **projection** columns is stored in letter cases specified by users and information in other columns is stored in uppercase.

12.2.14.75 STATEMENT_HISTORY

Displays information about statements executed on the current node. To query this system catalog, you must have the sysadmin permission. The result can be queried only in the system database but cannot be queried in the user database.

The constraints on the query of this system catalog are as follows:

- Data must be queried in the Postgres database. No data exists in other databases.
- This system catalog is controlled by **track_stmt_stat_level**. The default value is **OFF,L0**, where the first part controls full SQL statements, and the second part controls slow SQL statements. For details about the record level of each column, see the following table. To ensure system performance, you are advised to use the SET statement to change the value of this parameter so that the parameter takes effect only for the current session.
- For slow SQL statements, if the value of **track_stmt_stat_level** is not **OFF** and the SQL execution time exceeds the value of **log_min_duration_statement**, the SQL statement is recorded as a slow SQL statement.

Table 12-123 STATEMENT_HISTORY columns

Name	Type	Description	Record Level
db_name	name	Database name.	L0
schema_name	name	Schema name.	L0
origin_node	integer	Node name.	L0
user_name	name	Username.	L0
application_name	text	Name of the application that sends a request.	L0
client_addr	text	IP address of the client that sends a request.	L0
client_port	integer	Port number of the client that sends a request.	L0
unique_query_id	bigint	ID of the normalized SQL statement.	L0
debug_query_id	bigint	ID of the unique SQL statement. Some statements are not unique. For example, the value of **debug_query_id** in the Parse packet, DCL statements, and TCL statements is **0**.	L0

Name	Type	Description	Record Level
query	text	Normalized SQL statement. When the **track_stmt_parameter** parameter is enabled, the complete SQL statement is displayed.	L0
start_time	timestamp with time zone	Time when a statement starts.	L0
finish_time	timestamp with time zone	Time when a statement ends.	L0
slow_sql_threshold	bigint	Standard for slow SQL statement execution.	L0
transaction_id	bigint	Transaction ID.	L0
thread_id	bigint	ID of an execution thread.	L0
session_id	bigint	Session ID of a user.	L0
n_soft_parse	bigint	Number of soft parsing times. The value of **n_soft_parse** plus the value of **n_hard_parse** may be greater than the value of **n_calls** because the number of subqueries is not counted in the value of **n_calls**.	L0
n_hard_parse	bigint	Number of hard parsing times. The value of **n_soft_parse** plus the value of **n_hard_parse** may be greater than the value of **n_calls** because the number of subqueries is not counted in the value of **n_calls**.	L0
query_plan	text	Statement execution plan.	L1(Full SQL) L0(Slow SQL)
n_returned_rows	bigint	Number of rows in the result set returned by the **SELECT** statement.	L0
n_tuples_fetched	bigint	Number of rows randomly scanned.	L0
n_tuples_returned	bigint	Number of rows sequentially scanned.	L0

Name	Type	Description	Record Level
n_tuples_inserted	bigint	Number of rows inserted.	L0
n_tuples_updated	bigint	Number of rows updated.	L0
n_tuples_deleted	bigint	Number of rows deleted.	L0
n_blocks_fetched	bigint	Number of buffer block access times.	L0
n_blocks_hit	bigint	Number of buffer block hits.	L0
db_time	bigint	Valid DB time, which is accumulated if multiple threads are involved (unit: μs).	L0
cpu_time	bigint	CPU time (unit: μs).	L0
execution_time	bigint	Execution time in the executor (unit: μs).	L0
parse_time	bigint	SQL parsing time (unit: μs).	L0
plan_time	bigint	SQL plan generation time (unit: μs).	L0
rewrite_time	bigint	SQL rewriting time (unit: μs).	L0
pl_execution_time	bigint	Execution time of PL/pgSQL (unit: μs).	L0
pl_compilation_time	bigint	Compilation time of PL/pgSQL (unit: μs).	L0
data_io_time	bigint	I/O time (unit: μs).	L0
net_send_info	text	Network status of messages sent through a physical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This column is not supported in the centralized system.	L0
net_recv_info	text	Network status of messages received through a physical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This column is not supported in the centralized system.	L0

Name	Type	Description	Record Level
net_stream_send_info	text	Network status of messages sent through a logical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This column is not supported in the centralized system.	L0
net_stream_recv_info	text	Network status of messages received through a logical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This column is not supported in the centralized system.	L0
lock_count	bigint	Number of locks.	L0
lock_time	bigint	Time required for locking.	L1
lock_wait_count	bigint	Number of lock waits.	L0
lock_wait_time	bigint	Time required for lock waiting.	L1
lock_max_count	bigint	Maximum number of locks.	L0
lwlock_count	bigint	Number of lightweight locks (reserved).	L0
lwlock_wait_count	bigint	Number of lightweight lock waits.	L0
lwlock_time	bigint	Time required for lightweight locking (reserved).	L1
lwlock_wait_time	bigint	Time required for lightweight lock waiting.	L1

Name	Type	Description	Record Level
details	bytea	List of wait events and statement lock events. When the value of the record level is greater than or equal to L0, the list of wait events starts to be recorded. Statistics about the wait events of the current statement are displayed. For details about events, see **Table 12-412**, **Table 12-413**, **Table 12-414**, and **Table 12-415**. For details about the impact of each transaction lock on services, see **LOCK**. When the value of **track_stmt_stat_level** is **L2**, the list of statement lock events starts to be recorded. The list records events in chronological order. The number of records is determined by the **track_stmt_details_size** parameter. This column is in binary format and needs to be read by using the parsing function **pg_catalog.statement_detail_decode**. For details, see **Table 7-102**. Events include: • Start locking. • Complete locking. • Start lock waiting. • Complete lock waiting. • Start unlocking. • Complete unlocking. • Start lightweight lock waiting. • Complete lightweight lock waiting.	L0/L2
is_slow_sql	Boolean	Specifies whether the SQL statement is a slow SQL statement. • **t** (true): yes • **f** (false): no	L0
trace_id	text	Driver-specific trace ID, which is associated with an application request.	L0

Name	Type	Description	Record Level
advise	text	Risk information that may cause the SQL statement to be a slow SQL statement.	L0
parent_unique_sql_id	bigint	Normalized SQL ID of the outer SQL statement. For statements executed in a stored procedure, the value is the normalized SQL ID of the statement that invokes the stored procedure. For statements outside the stored procedure, the value is **0**.	L0
finish_status	text	Statement completion status. • **normal** (default) • **cancelled**: cancellation or termination.	L0

NOTICE

- The record level of the **query_plan** column is L1 for full SQL statements and L0 for slow SQL statements.
- For the **db_time** time model, the time statistics of each dimension in **statement_history** meet the requirement that **db_time** is greater than or equal to **max(cpu_time, parse_time, plan_time, rewrite_time, data_io_time, net_send_info.time, net_recv_info.time, net_stream_send_info.time, net_stream_recv_info.time)**.
- For the **db_time** time model formula, the following dimensions are not included in the statistical accuracy scope:
  1. **execution_time**
  2. **pl_execution_time**
  3. **pl_compilation_time**
- The maximum nesting depth of wait events is **20**. If the nesting depth exceeds 20, a new wait event overwrites the last wait event.

12.2.14.76 STREAMING_STREAM

STREAMING_STREAM records the metadata of all STREAM objects.

Table 12-124 STREAMING_STREAM columns

Name	Type	Description
relid	oid	STREAM object ID.

Name	Type	Description
queries	bytea	Bitmap mapping of the CONTVIEW corresponding to the STREAM.

12.2.14.77 STREAMING_CONT_QUERY

STREAMING_CONT_QUERY records the metadata of all CONTVIEW objects.

Table 12-125 STREAMING_CONT_QUERY columns

Name	Type	Description
id	integer	Unique identifier of the CONTVIEW object.
type	"char"	CONTVIEW type. ● **'r'**: CONTVIEW is based on the row-store model.
relid	oid	CONTVIEW object ID.
defrelid	oid	ID of the continuous computing rule view corresponding to CONTVIEW.
active	Boolean	Determines whether the CONTVIEW is in the continuous computing state. ● **t** (true): yes ● **f** (false): no
streamrelid	oid	ID of STREAM corresponding to CONTVIEW.
matrelid	oid	ID of the materialized table corresponding to CONTVIEW.
lookupidxid	oid	ID of GROUP LOOK UP INDEX corresponding to CONTVIEW. This column is for internal use and is available only in row-store tables.
step_factor	smallint	CONTVIEW step mode. The main values are **0** (no overlapping window) and **1** (sliding window, with one step).
ttl	integer	Value of **ttl_interval** set by CONTVIEW.
ttl_attno	smallint	Number of a time column corresponding to the TTL function set by CONTVIEW.
dictrelid	oid	ID of the dictionary table corresponding to CONTVIEW.
grpnum	smallint	Number of dimension columns in the CONTVIEW continuous computing rule. This column is for internal use.

Name	Type	Description
grpidx	int2vector	Index of the dimension column in TARGET LIST in the CONTVIEW continuous computing rule. This column is for internal use.

12.3 System Views

12.3.1 Partitioned Table

12.3.1.1 ADM_IND_PARTITIONS

ADM_IND_PARTITIONS displays the partition information about local indexes of level-1 partitioned table in the database (excluding global indexes on partitioned tables). Each local index partition of level-1 index partitioned table in the database, if present, has a row of records in ADM_IND_PARTITIONS. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-126 ADM_IND_PARTITIONS columns

Name	Type	Description
index_owner	character varying(64)	Owner name of the partitioned table index to which an index partition belongs
index_name	character varying(64)	Index name of the partitioned table index to which an index partition belongs
partition_name	character varying(64)	Name of an index partition
def_tablespace_name	name	Tablespace name of an index partition

Name	Type	Description
high_value	text	Upper limit of the partition corresponding to the index partition **NOTE** • For range partitioning and interval partitioning, the upper limit of each partition is displayed. • For list partitioning, the value list of each partition is displayed. • For hash partitioning, the number of each partition is displayed.
index_partition_usable	Boolean	Specifies whether an index partition is available: • **t** (true): yes • **f** (false): no
schema	character varying(64)	Schema of the partitioned table index to which an index partition belongs
high_value_length	integer	Character length of the limit of the partition corresponding to an index partition
composite	character varying(3)	Specifies whether the index is a local index on the level-2 partitioned table. This table does not store level-2 partition information. Therefore, the value is **NO**.
subpartition_count	numeric	Number of level-2 partitions in a partition. The value is **0** because this table does not store level-2 partition information.
partition_position	numeric	Position of an index partition in the index

Name	Type	Description
status	character varying(8)	Specifies whether the index partition is available.
tablespace_name	name	Name of the tablespace that contains the partition
pct_free	numeric	Minimum percentage of available space in a block
ini_trans	numeric	Initial number of transactions. The default value is **4**. The value is **NULL** for non-USTORE partitioned tables.
max_trans	numeric	Maximum number of transactions. The default value is **128**. The value is **NULL** for non-USTORE partitioned tables.
initial_extent	numeric	Not supported. Set it to **NULL**.
next_extent	numeric	Not supported. Set it to **NULL**.
min_extent	numeric	Not supported. Set it to **NULL**.
max_extent	numeric	Not supported. Set it to **NULL**.
max_size	numeric	Not supported. Set it to **NULL**.
pct_increase	numeric	Not supported. Set it to **NULL**.
freelists	numeric	Not supported. Set it to **NULL**.
freelist_groups	numeric	Not supported. Set it to **NULL**.
logging	character varying(7)	Specifies whether changes to an index are recorded.

Name	Type	Description
compression	character varying(13)	Specifies whether an index compression is enabled for a partitioned index.
blevel	numeric	Not supported. Set it to **NULL**.
leaf_blocks	numeric	Not supported. Set it to **NULL**.
distinct_keys	numeric	Not supported. Set it to **NULL**.
avg_leaf_blocks_per_key	numeric	Not supported. Set it to **NULL**.
avg_data_blocks_per_key	numeric	Not supported. Set it to **NULL**.
clustering_factor	numeric	Sequence of a row in the table based on the index value You need to run the analyze command to collect statistics.
num_rows	numeric	Number of rows in a partition You need to run the vacuum command to collect statistics.
sample_size	numeric	Not supported. Set it to **NULL**.
last_analyzed	timestamp with time zone	Last date when the partition was analyzed
buffer_pool	character varying(7)	Actual buffer pool of a partition
flash_cache	character varying(7)	Not supported. Set it to **NULL**.
cell_flash_cache	character varying(7)	Not supported. Set it to **NULL**.
user_stats	character varying(3)	Not supported. Set it to **NULL**.
pct_direct_access	numeric	Not supported. Set it to **NULL**.
global_stats	character varying(3)	Not supported. Set it to **NULL**.

Name	Type	Description
domidx_opstatus	character varying(6)	Not supported. Set it to **NULL**.
parameters	character varying(1000)	Not supported. Set it to **NULL**.
interval	character varying(3)	Specifies whether the partition is a partition in an interval partitioned table.
segment_created	character varying(3)	Specifies whether an index partition segment has been created.
orphaned_entries	character varying(3)	Not supported. Set it to **NULL**.

12.3.1.2 ADM_IND_SUBPARTITIONS

ADM_IND_SUBPARTITIONS displays the partition information about local indexes of level-2 partitioned table in the database (excluding global indexes on partitioned tables). Each local index partition of level-2 index partitioned table in the database, if present, has a row of records in ADM_IND_SUBPARTITIONS. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG_CATALOG and SYS schema.

Table 12-127 ADM_IND_SUBPARTITIONS columns

Name	Type	Description
index_owner	character varying(64)	Name of the owner of the partitioned table index to which an index partition belongs.
index_name	character varying(64)	Name of the partitioned table index to which the index partition belongs.
partition_name	character varying(64)	Name of the level-1 partition where an index is located.
subpartition_name	character varying(64)	Name of the level-2 partition where an index is located.
def_tablespace_name	name	Tablespace name of the index partition.

Name	Type	Description
high_value	text	Limit of the partition corresponding to an index partition. • For range partitioning and interval partitioning, the upper limit of each partition is displayed. • For list partitioning, the value list of each partition is displayed. • For hash partitioning, the number of each partition is displayed.
index_partition_usable	Boolean	Specifies whether an index partition is available. • **t** (true): yes. • **f** (false): no.
schema	character varying(64)	Schema of the partitioned table index to which an index partition belongs.
high_value_length	integer	Character length of the limit of the partition corresponding to an index partition.
partition_position	numeric	Position of an index partition in the index.
subpartition_position	numeric	Position of the level-2 partition in the partition.
status	character varying(8)	Specifies whether an index partition is available.
tablespace_name	name	Tablespace name of the index partition.
pct_free	numeric	Percentage of minimum available space in a block.

Name	Type	Description
ini_trans	numeric	Initial number of transactions. The default value is **4**. The value is **NULL** for non-Ustore partitioned tables.
max_trans	numeric	Maximum number of transactions. The default value is **128**. The value is **NULL** for non-Ustore partitioned tables.
initial_extent	numeric	Not supported. Its value is **NULL**.
next_extent	numeric	Not supported. Its value is **NULL**.
min_extent	numeric	Not supported. Its value is **NULL**.
max_extent	numeric	Not supported. Its value is **NULL**.
max_size	numeric	Not supported. Its value is **NULL**.
pct_increase	numeric	Not supported. Its value is **NULL**.
freelists	numeric	Not supported. Its value is **NULL**.
freelist_groups	numeric	Not supported. Its value is **NULL**.
logging	character varying(7)	Specifies whether changes to an index are logged.
compression	character varying(13)	Compression type used for level-2 partitions.
blevel	numeric	Not supported. Its value is **NULL**.
leaf_blocks	numeric	Not supported. Its value is **NULL**.
distinct_keys	numeric	Not supported. Its value is **NULL**.
avg_leaf_blocks_per_key	numeric	Not supported. Its value is **NULL**.

Name	Type	Description
avg_data_blocks_per_key	numeric	Not supported. Its value is **NULL**.
clustering_factor	numeric	Sequence of a row in the table based on the value of the index. You need to run the **analyze** command to collect statistics.
num_rows	numeric	Number of rows in a level-2 partition. You need to run the vacuum command to collect statistics.
sample_size	numeric	Not supported. Its value is **NULL**.
last_analyzed	timestamp with time zone	Last date when the partition was analyzed.
buffer_pool	character varying(7)	Buffer pool of a level-2 partition.
flash_cache	character varying(7)	Not supported. Its value is **NULL**.
cell_flash_cache	character varying(7)	Not supported. Its value is **NULL**.
user_stats	character varying(3)	Not supported. Its value is **NULL**.
global_stats	character varying(3)	Not supported. Its value is **NULL**.
interval	character varying(3)	Specifies whether the partition is a partition in an interval partitioned table.
segment_created	character varying(3)	Specifies whether an index partition segment has been created.
domidx_opstatus	character varying(6)	Not supported. Its value is **NULL**.
parameters	character varying(1000)	Not supported. Its value is **NULL**.

12.3.1.3 ADM_PART_COL_STATISTICS

ADM_PART_COL_STATISTICS displays the column statistics and histogram information about all table partitions in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-128 ADM_PART_COL_STATISTICS columns

Name	Type	Description
owner	character varying(128)	Owner of the partitioned table
table_name	character varying(128)	Table name.
partition_name	character varying(128)	Table partition name
column_name	character varying(4000)	Column name.
num_distinct	numeric	Not supported. Set it to **NULL**.
low_value	raw	Not supported. Set it to **NULL**.
high_value	raw	Not supported. Set it to **NULL**.
density	numeric	Not supported. Set it to **NULL**.
num_nulls	numeric	Not supported. Set it to **NULL**.
num_buckets	numeric	Not supported. Set it to **NULL**.
sample_size	numeric	Not supported. Set it to **NULL**.
last_analyzed	timestamp(0) without time zone	Not supported. Set it to **NULL**.
global_stats	character varying(3)	Not supported. Set it to **NULL**.
user_stats	character varying(3)	Not supported. Set it to **NULL**.
notes	character varying(63)	Not supported. Set it to **NULL**.
avg_col_len	numeric	Not supported. Set it to **NULL**.
histogram	character varying(15)	Not supported. Set it to **NULL**.

Name	Type	Description
schema	character varying(64)	Name of the namespace to which the column belongs.

12.3.1.4 ADM_PART_INDEXES

ADM_PART_INDEXES displays information about all partitioned table indexes (excluding global indexes of partitioned tables) in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG_CATALOG and SYS schema.

Table 12-129 ADM_PART_INDEXES columns

Name	Type	Description
def_tablespace_name	name	Tablespace name of a partitioned table index.
index_owner	character varying(64)	Owner name of a partitioned table index.
index_name	character varying(64)	Name of a partitioned table index.
partition_count	bigint	Number of index partitions of a partitioned table index.
partitioning_key_count	integer	Number of partition keys of a partitioned table.
partitioning_type	text	Partitioning policy of a partitioned table. **NOTE** For details about the supported partitioning policies of the current partitioned table, see **CREATE TABLE PARTITION**.
schema	character varying(64)	Name of the schema to which a partitioned table index belongs.
table_name	character varying(64)	Name of the partitioned table to which a partitioned table index belongs.

Name	Type	Description
subpartitioning_type	text	Partitioning policy of a level-2 partitioned table. If the partitioned table is a level-1 partitioned table, **NONE** is displayed. **NOTE** For details about the supported partitioning policies of the current level-2 partitioned table, see **CREATE TABLE SUBPARTITION**.
def_subpartition_count	integer	Default number of level-2 partitions to be created. The value is **1** for level-2 partitioned tables and **0** for level-1 partitioned tables.
subpartitioning_key_count	integer	Number of level-2 partition keys of the partitioned table.

12.3.1.5 ADM_PART_TABLES

ADM_PART_TABLES displays information about all partitioned tables in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG_CATALOG and SYS schema.

Table 12-130 ADM_PART_TABLES columns

Name	Type	Description
table_owner	character varying(64)	Owner name of a partitioned table.
table_name	character varying(64)	Name of a partitioned table.
partitioning_type	text	Partitioning policy of a partitioned table. **NOTE** For details about the supported partitioning policies of the current partitioned table, see **CREATE TABLE PARTITION**.
partition_count	bigint	Number of partitions of a partitioned table.
partitioning_key_count	integer	Number of partition keys of a partitioned table.

Name	Type	Description
def_tablespace_name	name	Tablespace name of a partitioned table.
schema	character varying(64)	Schema of a partitioned table.
subpartitioning_type	text	Partitioning policy of a level-2 partitioned table. If the partitioned table is a level-1 partitioned table, **NONE** is displayed. **NOTE** For details about the supported partitioning policies of the current level-2 partitioned table, see **CREATE TABLE SUBPARTITION**.
def_subpartition_count	integer	Default number of level-2 partitions to be created. The value is **1** for a level-2 partitioned table and **0** for a level-1 partitioned table.
subpartitioning_key_count	integer	Number of level-2 partition keys of the partitioned table.
status	character varying(8)	Not supported. The value is **valid**.
def_pct_free	numeric	Default value of **PCTFREE** used when a partition is added.
def_pct_used	numeric	Not supported. Its value is **NULL**.
def_ini_trans	numeric	Default value of **INITRANS** used when a partition is added.
def_max_trans	numeric	Default value of **MAXTRANS** used when a partition is added.
def_initial_extent	character varying(40)	Not supported. Its value is **NULL**.
def_next_extent	character varying(40)	Not supported. Its value is **NULL**.
def_min_extents	character varying(40)	Not supported. Its value is **NULL**.

Name	Type	Description
def_max_extents	character varying(40)	Not supported. Its value is **NULL**.
def_max_size	character varying(40)	Not supported. Its value is **NULL**.
def_pct_increase	character varying(40)	Not supported. Its value is **NULL**.
def_freelists	numeric	Not supported. Its value is **NULL**.
def_freelist_groups	numeric	Not supported. Its value is **NULL**.
def_logging	character varying(7)	Not supported. Its value is **NULL**.
def_compression	character varying(8)	Default compression mode used when a partition is added. The options are as follows: • **NONE** • **ENABLED** • **DISABLED**
def_compress_for	character varying(30)	Default compression mode used when a partition is added. **NOTE** For available compression modes and compression levels, see **WITH ({ storage_parameter = value } [, ...])**.
def_buffer_pool	character varying(7)	Not supported. The value is **DEFAULT**.
def_flash_cache	character varying(7)	Not supported. Its value is **NULL**.
def_cell_flash_cache	character varying(7)	Not supported. Its value is **NULL**.
ref_ptn_constraint_name	character varying(128)	Not supported. Its value is **NULL**.
interval	character varying(1000)	Interval.

Name	Type	Description
autolist	text	Specifies whether to enable the automatic extension function for the level-1 list partition. • **YES**: Enable the automatic extension function. • **NO**: Disable the automatic extension function.
interval_subpartition	character varying(1000)	Not supported. Its value is **NULL**.
autolist_subpartition	text	Specifies whether to enable the automatic extension function for the level-2 list partition. • **YES**: Enable the automatic extension function. • **NO**: Disable the automatic extension function.
is_nested	character varying(3)	Not supported. The value is **NO**.
def_segment_creation	character varying(4)	Currently, the segment-page mode is not supported. When the segment-page mode is enabled, the value is **YES**.
def_indexing	character varying(3)	Not supported. The value is **ON**.
def_inmemory	character varying(8)	Not supported. The value is **NONE**.
def_inmemory_priority	character varying(8)	Not supported. Its value is **NULL**.
def_inmemory_distribute	character varying(15)	Not supported. Its value is **NULL**.
def_inmemory_compression	character varying(17)	Not supported. Its value is **NULL**.
def_inmemory_duplicate	character varying(13)	Not supported. Its value is **NULL**.
def_read_only	character varying(3)	Not supported. The value is **NO**.
def_cellmemory	character varying(24)	Not supported. Its value is **NULL**.

Name	Type	Description
def_inmemory_service	character varying(12)	Not supported. Its value is **NULL**.
def_inmemory_service_name	character varying(1000)	Not supported. Its value is **NULL**.

12.3.1.6 ADM_SUBPART_COL_STATISTICS

ADM_SUBPART_COL_STATISTICS displays the column statistics and histogram information about all level-2 partitions in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-131 ADM_SUBPART_COL_STATISTICS columns

Name	Type	Description
owner	character varying(128)	Table owner.
table_name	character varying(128)	Table name.
subpartition_name	character varying(128)	Name of a level-2 partition.
column_name	character varying(4000)	Column name.
num_distinct	numeric	Not supported. Set it to **NULL**.
low_value	raw	Not supported. Set it to **NULL**.
high_value	raw	Not supported. Set it to **NULL**.
density	numeric	Not supported. Set it to **NULL**.
num_nulls	numeric	Not supported. Set it to **NULL**.
num_buckets	numeric	Not supported. Set it to **NULL**.
sample_size	numeric	Not supported. Set it to **NULL**.
last_analyzed	timestamp(0) without time zone	Not supported. Set it to **NULL**.
global_stats	character varying(3)	Not supported. Set it to **NULL**.

Name	Type	Description
user_stats	character varying(3)	Not supported. Set it to **NULL**.
notes	character varying(41)	Not supported. Set it to **NULL**.
avg_col_len	numeric	Not supported. Set it to **NULL**.
histogram	character varying(15)	Not supported. Set it to **NULL**.
schema	character varying(64)	Name of the namespace to which the column belongs.

12.3.1.7 ADM_SUBPART_KEY_COLUMNS

ADM_SUBPART_KEY_COLUMNS displays information about the partition key columns of level-2 partitioned tables or partitioned indexes in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG_CATALOG and SYS schema.

Table 12-132 ADM_SUBPART_KEY_COLUMNS columns

Name	Type	Description
owner	character varying(128)	Owner of a level-2 partitioned table or index.
name	character varying(128)	Name of a level-2 partitioned table or index.
object_type	character(5)	Object type. • Its value is **table** for a level-2 partitioned table. • Its value is **index** for a level-2 partitioned index.
column_name	character varying(4000)	Partition key column name of a level-2 partitioned table or index.
column_position	numeric	Position of a column in a partition.
collated_column_id	numeric	Not supported. Its value is **NULL**.

12.3.1.8 ADM_TAB_SUBPARTITIONS

ADM_TAB_SUBPARTITIONS displays information about all level-2 partitions in the database. By default, only the system administrator can access this view. Common

users can access the view only after being authorized. This view exists in both PG_CATALOG and SYS schema.

Table 12-133 ADM_TAB_SUBPARTITIONS columns

Name	Type	Description
table_owner	character varying(64)	Table owner.
table_name	character varying(64)	Relational table name.
partition_name	character varying(64)	Partition name.
subpartition_name	character varying(64)	Name of a level-2 partition.
high_value	text	Limit of a level-2 partition. • For range partitioning and interval partitioning, the upper limit of each partition is displayed. • For list partitioning, the value list of each partition is displayed. • For hash partitioning, the number of each partition is displayed.
tablespace_name	name	Tablespace name of a level-2 partitioned table.
schema	character varying(64)	Name of a namespace.
high_value_length	integer	Character length of the limit of a level-2 partition.

12.3.1.9 ADM_TAB_PARTITIONS

ADM_TAB_PARTITIONS displays information about level-1 partitions (including level-2 partitioned tables) in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG_CATALOG and SYS schema.

Table 12-134 ADM_TAB_PARTITIONS columns

Name	Type	Description
table_owner	character varying(64)	Table owner.
table_name	character varying(64)	Relational table name.
partition_name	character varying(64)	Partition name.

Name	Type	Description
high_value	text	Limit of a partition. • For range partitioning and interval partitioning, the upper limit of each partition is displayed. • For list partitioning, the value list of each partition is displayed. • For hash partitioning, the number of each partition is displayed.
tablespace_name	name	Tablespace name of a partitioned table.
schema	character varying(64)	Name of a namespace.
subpartition_count	bigint	Number of level-2 partitions.
high_value_length	integer	Length of the partition boundary value expression.
composite	character varying(3)	Specifies whether the table is a level-2 partitioned table.
partition_position	numeric	Position of the partition in the table.
pct_free	numeric	Minimum percentage of available space in a block.
pct_used	numeric	Not supported. Its value is **NULL**.
ini_trans	numeric	Initial number of transactions. The default value is **4**. The value is **NULL** for non-Ustore partitioned tables.
max_trans	numeric	Maximum number of transactions. The default value is **128**. The value is **NULL** for non-Ustore partitioned tables.
initial_extent	numeric	Not supported. Its value is **NULL**.
next_extent	numeric	Not supported. Its value is **NULL**.
min_extent	numeric	Not supported. Its value is **NULL**.
max_extent	numeric	Not supported. Its value is **NULL**.
max_size	numeric	Not supported. Its value is **NULL**.

Name	Type	Description
pct_increase	numeric	Not supported. Its value is **NULL**.
freelists	numeric	Not supported. Its value is **NULL**.
freelist_groups	numeric	Not supported. Its value is **NULL**.
logging	character varying(7)	Specifies whether changes to a table are logged.
compression	character varying(8)	Actual compression attribute of a partitioned table.
compress_for	character varying(30)	Not supported. Its value is **NULL**.
num_rows	numeric	Number of rows in a partition.
blocks	numeric	Not supported. Its value is **NULL**.
empty_blocks	numeric	Not supported. Its value is **NULL**.
avg_space	numeric	Not supported. Its value is **NULL**.
chain_cnt	numeric	Not supported. Its value is **NULL**.
avg_row_len	numeric	Not supported. Its value is **NULL**.
sample_size	numeric	Not supported. Its value is **NULL**.
last_analyzed	timestamp with time zone	Last date when the partition was analyzed.
buffer_pool	character varying(7)	Buffer pool allocated to a partitioned block.
flash_cache	character varying(7)	Not supported. Its value is **NULL**.
cell_flash_cache	character varying(7)	Not supported. Its value is **NULL**.
global_stats	character varying(3)	Not supported. Its value is **NULL**.
user_stats	character varying(3)	Not supported. Its value is **NULL**.
is_nested	character varying(3)	Specifies whether this partitioned table is a nested partitioned table.
parent_table_partition	character varying(128)	Not supported. Its value is **NULL**.
interval	character varying(3)	Specifies whether the partition is a partition in an interval partitioned table.
segment_created	character varying(4)	Specifies whether a table partition has segments created.
indexing	character varying(4)	Not supported. Its value is **NULL**.

Name	Type	Description
read_only	character varying(4)	Not supported. Its value is **NULL**.
inmemory	character varying(8)	Not supported. Its value is **NULL**.
inmemory_priority	character varying(8)	Not supported. Its value is **NULL**.
inmemory_distribute	character varying(15)	Not supported. Its value is **NULL**.
inmemory_compression	character varying(17)	Not supported. Its value is **NULL**.
inmemory_duplicate	character varying(13)	Not supported. Its value is **NULL**.
cellmemory	character varying(24)	Not supported. Its value is **NULL**.
inmemory_service	character varying(12)	Not supported. Its value is **NULL**.
inmemory_service_name	character varying(100)	Not supported. Its value is **NULL**.
memoptimize_read	character varying(8)	Not supported. Its value is **NULL**.
memoptimize_write	character varying(8)	Not supported. Its value is **NULL**.

12.3.1.10 DB_IND_PARTITIONS

DB_IND_PARTITIONS displays partition information about local indexes of level-1 partitioned table accessible to the current user (excluding global indexes of partitioned tables). It is accessible to all users. This view exists in both PG_CATALOG and SYS schema.

Table 12-135 DB_IND_PARTITIONS columns

Name	Type	Description
index_owner	character varying(64)	Name of the owner of the partitioned table index to which an index partition belongs.
index_name	character varying(64)	Name of the partitioned table index to which the index partition belongs.
partition_name	character varying(64)	Name of an index partition.

Name	Type	Description
def_tablespace_name	name	Tablespace name of the index partition.
high_value	text	Upper limit of the partition corresponding to the index partition.
index_partition_usable	Boolean	Specifies whether an index partition is available. • **t** (true): yes. • **f** (false): no.
schema	character varying(64)	Schema of the partitioned table index to which an index partition belongs.
high_value_length	integer	Character length of the limit of the partition corresponding to an index partition.
composite	character varying(3)	Specifies whether the index is a local index on the level-2 partitioned table. This table does not store level-2 partition information. Therefore, the value is **NO**.
subpartition_count	numeric	Number of level-2 partitions in a partition. This table does not store level-2 partition information. Therefore, the value is **0**.
partition_position	numeric	Position of an index partition in the index.
status	character varying(8)	Specifies whether an index partition is available.
tablespace_name	name	Name of the tablespace that contains the partition.
pct_free	numeric	Percentage of minimum available space in a block.

Name	Type	Description
ini_trans	numeric	Initial number of transactions. The default value is **4**. The value is **NULL** for non-Ustore partitioned tables.
max_trans	numeric	Maximum number of transactions. The default value is **128**. The value is **NULL** for non-Ustore partitioned tables.
initial_extent	numeric	Not supported. Its value is **NULL**.
next_extent	numeric	Not supported. Its value is **NULL**.
min_extent	numeric	Not supported. Its value is **NULL**.
max_extent	numeric	Not supported. Its value is **NULL**.
max_size	numeric	Not supported. Its value is **NULL**.
pct_increase	numeric	Not supported. Its value is **NULL**.
freelists	numeric	Not supported. Its value is **NULL**.
freelist_groups	numeric	Not supported. Its value is **NULL**.
logging	character varying(7)	Specifies whether changes to an index are logged.
compression	character varying(13)	Specifies whether an index compression is enabled for a partitioned index.
blevel	numeric	Not supported. Its value is **NULL**.
leaf_blocks	numeric	Not supported. Its value is **NULL**.
distinct_keys	numeric	Not supported. Its value is **NULL**.

Name	Type	Description
avg_leaf_blocks_per_key	numeric	Not supported. Its value is **NULL**.
avg_data_blocks_per_key	numeric	Not supported. Its value is **NULL**.
clustering_factor	numeric	Sequence of a row in the table based on the value of the index. You need to run the **analyze** command to collect statistics.
num_rows	numeric	Number of rows in a partition. You need to run the vacuum command to collect statistics.
sample_size	numeric	Not supported. Its value is **NULL**.
last_analyzed	timestamp with time zone	Last date when the partition was analyzed.
buffer_pool	character varying(7)	Actual buffer pool of a partition.
flash_cache	character varying(7)	Not supported. Its value is **NULL**.
cell_flash_cache	character varying(7)	Not supported. Its value is **NULL**.
user_stats	character varying(3)	Not supported. Its value is **NULL**.
pct_direct_access	numeric	Not supported. Its value is **NULL**.
global_stats	character varying(3)	Not supported. Its value is **NULL**.
domidx_opstatus	character varying(6)	Not supported. Its value is **NULL**.
parameters	character varying(1000)	Not supported. Its value is **NULL**.
interval	character varying(3)	Specifies whether the partition is a partition in an interval partitioned table.

Name	Type	Description
segment_created	character varying(3)	Specifies whether an index partition segment has been created.
orphaned_entries	character varying(3)	Not supported. Its value is **NULL**.

12.3.1.11 DB_IND_SUBPARTITIONS

DB_IND_SUBPARTITIONS displays partition information about local indexes of level-2 partitioned table accessible to the current user (excluding global indexes of partitioned tables). It is accessible to all users. This view exists in both PG_CATALOG and SYS schema.

Table 12-136 DB_IND_SUBPARTITIONS columns

Name	Type	Description
index_owner	character varying(64)	Name of the owner of the partitioned table index to which an index partition belongs.
index_name	character varying(64)	Name of the partitioned table index to which the index partition belongs.
partition_name	character varying(64)	Name of the level-1 partition where an index is located.
subpartition_name	character varying(64)	Name of the level-2 partition where an index is located.
def_tablespace_name	name	Tablespace name of the index partition.
high_value	text	Limit of the partition corresponding to an index partition.
index_partition_usable	Boolean	Specifies whether an index partition is available. • **t** (true): yes. • **f** (false): no.

Name	Type	Description
schema	character varying(64)	Schema of the partitioned table index to which an index partition belongs.
high_value_length	integer	Character length of the limit of the partition corresponding to an index partition.
partition_position	numeric	Position of an index partition in the index.
subpartition_position	numeric	Position of the level-2 partition in the partition.
status	character varying(8)	Specifies whether an index partition is available.
tablespace_name	name	Tablespace name of the index partition.
pct_free	numeric	Percentage of minimum available space in a block.
ini_trans	numeric	Initial number of transactions. The default value is **4**. The value is **NULL** for non-Ustore partitioned tables.
max_trans	numeric	Maximum number of transactions. The default value is **128**. The value is **NULL** for non-Ustore partitioned tables.
initial_extent	numeric	Not supported. Its value is **NULL**.
next_extent	numeric	Not supported. Its value is **NULL**.
min_extent	numeric	Not supported. Its value is **NULL**.
max_extent	numeric	Not supported. Its value is **NULL**.
max_size	numeric	Not supported. Its value is **NULL**.

Name	Type	Description
pct_increase	numeric	Not supported. Its value is **NULL**.
freelists	numeric	Not supported. Its value is **NULL**.
freelist_groups	numeric	Not supported. Its value is **NULL**.
logging	character varying(7)	Specifies whether changes to an index are logged.
compression	character varying(13)	Compression type used for level-2 partitions.
blevel	numeric	Not supported. Its value is **NULL**.
leaf_blocks	numeric	Not supported. Its value is **NULL**.
distinct_keys	numeric	Not supported. Its value is **NULL**.
avg_leaf_blocks_per_key	numeric	Not supported. Its value is **NULL**.
avg_data_blocks_per_key	numeric	Not supported. Its value is **NULL**.
clustering_factor	numeric	Sequence of a row in the table based on the value of the index. You need to run the **analyze** command to collect statistics.
num_rows	numeric	Number of rows in a level-2 partition. You need to run the vacuum command to collect statistics.
sample_size	numeric	Not supported. Its value is **NULL**.
last_analyzed	timestamp with time zone	Last date when the partition was analyzed.
buffer_pool	character varying(7)	Buffer pool of a level-2 partition.
flash_cache	character varying(7)	Not supported. Its value is **NULL**.

Name	Type	Description
cell_flash_cache	character varying(7)	Not supported. Its value is **NULL**.
user_stats	character varying(3)	Not supported. Its value is **NULL**.
global_stats	character varying(3)	Not supported. Its value is **NULL**.
interval	character varying(3)	Specifies whether the partition is a partition in an interval partitioned table.
segment_created	character varying(3)	Specifies whether an index partition segment has been created.
domidx_opstatus	character varying(6)	Not supported. Its value is **NULL**.
parameters	character varying(1000)	Not supported. Its value is **NULL**.

12.3.1.12 DB_PART_COL_STATISTICS

DB_PART_COL_STATISTICS displays column statistics and histogram information about table partitions accessible to the current user. All users can access this view. This view exists in the PG_CATALOG and SYS schemas.

Table 12-137 DB_PART_COL_STATISTICS columns

Name	Type	Description
owner	character varying(128)	Owner of the partitioned table
table_name	character varying(128)	Table name.
partition_name	character varying(128)	Table partition name
column_name	character varying(4000)	Column name.
num_distinct	numeric	Not supported. Set it to **NULL**.
low_value	raw	Not supported. Set it to **NULL**.
high_value	raw	Not supported. Set it to **NULL**.

Name	Type	Description
density	numeric	Not supported. Set it to **NULL**.
num_nulls	numeric	Not supported. Set it to **NULL**.
num_buckets	numeric	Not supported. Set it to **NULL**.
sample_size	numeric	Not supported. Set it to **NULL**.
last_analyzed	timestamp(0) without time zone	Not supported. Set it to **NULL**.
global_stats	character varying(3)	Not supported. Set it to **NULL**.
user_stats	character varying(3)	Not supported. Set it to **NULL**.
notes	character varying(63)	Not supported. Set it to **NULL**.
avg_col_len	numeric	Not supported. Set it to **NULL**.
histogram	character varying(15)	Not supported. Set it to **NULL**.
schema	character varying(64)	Name of the namespace to which the column belongs.

12.3.1.13 DB_PART_INDEXES

DB_PART_INDEXES displays information about partitioned table indexes (excluding global indexes of partitioned tables) accessible to the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-138 DB_PART_INDEXES columns

Name	Type	Description
def_tablespace_name	name	Tablespace name of a partitioned table index
index_owner	character varying(64)	Owner name of a partitioned table index
index_name	character varying(64)	Name of a partitioned table index

Name	Type	Description
partition_count	bigint	Number of index partitions of a partitioned table index
partitioning_key_count	integer	Number of partition keys of a partitioned table
partitioning_type	text	Partitioning policy of a partitioned table **NOTE** For details about the supported partitioning policies of the current partitioned table, see **CREATE TABLE PARTITION**.
schema	character varying(64)	Name of the schema to which a partitioned table index belongs
table_name	character varying(64)	Name of the partitioned table to which a partitioned table index belongs
subpartitioning_type	text	Partitioning policy of a level-2 partitioned table. If the partitioned table is a level-1 partitioned table, **NONE** is displayed. **NOTE** For details about the supported partitioning policies of the current level-2 partitioned table, see **CREATE TABLE SUBPARTITION**.
def_subpartition_count	integer	Default number of level-2 partitions to be created. The value is **1** for a level-2 partitioned table and **0** for a level-1 partitioned table.
subpartitioning_key_count	integer	Number of level-2 partition keys of a partitioned table

12.3.1.14 DB_PART_KEY_COLUMNS

DB_PART_KEY_COLUMNS displays information about the partition key columns of partitioned tables or partitioned indexes accessible to the current user. This view is accessible to all users and displays all information accessible to the current user. This view exists in both PG_CATALOG and SYS schema.

Table 12-139 DB_PART_KEY_COLUMNS columns

Name	Type	Description
owner	character varying(128)	Owner of a partitioned table or index.
name	character varying(128)	Name of a partitioned table or index.
object_type	character(5)	Object type. • Its value is **table** for a partitioned table. • Its value is **index** for a partitioned index.
column_name	character varying(4000)	Partition key column name of a partitioned table or index.
column_position	numeric	Position of a column in a partition.
collated_column_id	numeric	Not supported. Its value is **NULL**.

12.3.1.15 DB_PART_TABLES

DB_PART_TABLES displays information about partitioned tables accessible to the current user. This view exists in both PG_CATALOG and SYS schema.

Table 12-140 DB_PART_TABLES columns

Name	Type	Description
table_owner	character varying(64)	Owner name of a partitioned table.
table_name	character varying(64)	Name of a partitioned table.
partitioning_type	text	Partitioning policy of a partitioned table. **NOTE** For details about the supported partitioning policies of the current partitioned table, see **CREATE TABLE PARTITION**.
partition_count	bigint	Number of partitions of a partitioned table.
partitioning_key_count	integer	Number of partition keys of a partitioned table.

Name	Type	Description
def_tablespace_name	name	Tablespace name of a partitioned table.
schema	character varying(64)	Schema of a partitioned table.
subpartitioning_type	text	Partitioning policy of a level-2 partitioned table. If the partitioned table is a level-1 partitioned table, **NONE** is displayed. **NOTE** For details about the supported partitioning policies of the current level-2 partitioned table, see **CREATE TABLE SUBPARTITION**.
def_subpartition_count	integer	Default number of level-2 partitions to be created. The value is **1** for a level-2 partitioned table and **0** for a level-1 partitioned table.
subpartitioning_key_count	integer	Number of level-2 partition keys of the partitioned table.
status	character varying(8)	Not supported. The value is **valid**.
def_pct_free	numeric	Default value of **PCTFREE** used when a partition is added.
def_pct_used	numeric	Not supported. Its value is **NULL**.
def_ini_trans	numeric	Default value of **INITRANS** used when a partition is added.
def_max_trans	numeric	Default value of **MAXTRANS** used when a partition is added.
def_initial_extent	character varying(40)	Not supported. Its value is **NULL**.
def_next_extent	character varying(40)	Not supported. Its value is **NULL**.
def_min_extents	character varying(40)	Not supported. Its value is **NULL**.

Name	Type	Description
def_max_extents	character varying(40)	Not supported. Its value is **NULL**.
def_max_size	character varying(40)	Not supported. Its value is **NULL**.
def_pct_increase	character varying(40)	Not supported. Its value is **NULL**.
def_freelists	numeric	Not supported. Its value is **NULL**.
def_freelist_groups	numeric	Not supported. Its value is **NULL**.
def_logging	character varying(7)	Not supported. Its value is **NULL**.
def_compression	character varying(8)	Default compression mode used when a partition is added. The options are as follows: • **NONE** • **ENABLED** • **DISABLED**
def_compress_for	character varying(30)	Default compression mode used when a partition is added. **NOTE** For available compression modes and compression levels, see **WITH ({ storage_parameter = value } [, ...])**.
def_buffer_pool	character varying(7)	Not supported. The value is **DEFAULT**.
def_flash_cache	character varying(7)	Not supported. Its value is **NULL**.
def_cell_flash_cache	character varying(7)	Not supported. Its value is **NULL**.
ref_ptn_constraint_name	character varying(128)	Not supported. Its value is **NULL**.
interval	character varying(1000)	Interval.

Name	Type	Description
autolist	text	Specifies whether to enable the automatic extension function for the level-1 list partition. • **YES**: Enable the automatic extension function. • **NO**: Disable the automatic extension function.
interval_subpartition	character varying(1000)	Not supported. Its value is **NULL**.
autolist_subpartition	text	Specifies whether to enable the automatic extension function for the level-2 list partition. • **YES**: Enable the automatic extension function. • **NO**: Disable the automatic extension function.
is_nested	character varying(3)	Not supported. The value is **NO**.
def_segment_creation	character varying(4)	Currently, the segment-page mode is not supported. When the segment-page mode is enabled, the value is **YES**.
def_indexing	character varying(3)	Not supported. The value is **ON**.
def_inmemory	character varying(8)	Not supported. The value is **NONE**.
def_inmemory_priority	character varying(8)	Not supported. Its value is **NULL**.
def_inmemory_distribute	character varying(15)	Not supported. Its value is **NULL**.
def_inmemory_compression	character varying(17)	Not supported. Its value is **NULL**.
def_inmemory_duplicate	character varying(13)	Not supported. Its value is **NULL**.
def_read_only	character varying(3)	Not supported. The value is **NO**.
def_cellmemory	character varying(24)	Not supported. Its value is **NULL**.

Name	Type	Description
def_inmemory_service	character varying(12)	Not supported. Its value is **NULL**.
def_inmemory_service_name	character varying(1000)	Not supported. Its value is **NULL**.

12.3.1.16 DB_SUBPART_COL_STATISTICS

DB_SUBPART_COL_STATISTICS displays column statistics and histogram information about level-2 partitions of partitioned objects accessible to the current user. All users can access this view. This view exists in the PG_CATALOG and SYS schemas.

Table 12-141 DB_SUBPART_COL_STATISTICS columns

Name	Type	Description
owner	character varying(128)	Table owner.
table_name	character varying(128)	Table name.
subpartition_name	character varying(128)	Name of a level-2 partition.
column_name	character varying(4000)	Column name.
num_distinct	numeric	Not supported. Set it to **NULL**.
low_value	raw	Not supported. Set it to **NULL**.
high_value	raw	Not supported. Set it to **NULL**.
density	numeric	Not supported. Set it to **NULL**.
num_nulls	numeric	Not supported. Set it to **NULL**.
num_buckets	numeric	Not supported. Set it to **NULL**.
sample_size	numeric	Not supported. Set it to **NULL**.
last_analyzed	timestamp(0) without time zone	Not supported. Set it to **NULL**.
global_stats	character varying(3)	Not supported. Set it to **NULL**.

Name	Type	Description
user_stats	character varying(3)	Not supported. Set it to **NULL**.
notes	character varying(41)	Not supported. Set it to **NULL**.
avg_col_len	numeric	Not supported. Set it to **NULL**.
histogram	character varying(15)	Not supported. Set it to **NULL**.
schema	character varying(64)	Name of the namespace to which the column belongs.

12.3.1.17 DB_SUBPART_KEY_COLUMNS

DB_SUBPART_KEY_COLUMNS displays information about the partition key columns of level-2 partitioned tables or partitioned indexes accessible to the current user. This view is accessible to all users and displays all information accessible to the current user. This view exists in both PG_CATALOG and SYS schema.

Table 12-142 DB_SUBPART_KEY_COLUMNS columns

Name	Type	Description
owner	character varying(128)	Owner of a level-2 partitioned table or index.
name	character varying(128)	Name of a level-2 partitioned table or index.
object_type	character(5)	Object type. • Its value is **table** for a partitioned table. • Its value is **index** for a partitioned index.
column_name	character varying(4000)	Partition key column name of a level-2 partitioned table or index.
column_position	numeric	Position of a column in a partition.
collated_column_id	numeric	Not supported. Its value is **NULL**.

12.3.1.18 DB_TAB_PARTITIONS

DB_TAB_PARTITIONS displays information about level-1 partitions (including level-2 partitioned tables) accessible to the current user. It is accessible to all users. This view exists in both PG_CATALOG and SYS schema.

Table 12-143 DB_TAB_PARTITIONS columns

Name	Type	Description
table_owner	character varying(64)	Table owner.
table_name	character varying(64)	Relational table name.
partition_name	character varying(64)	Partition name.
high_value	text	Limit of a partition. • For range partitioning and interval partitioning, the upper limit of each partition is displayed. • For list partitioning, the value list of each partition is displayed. • For hash partitioning, the number of each partition is displayed.
tablespace_name	name	Tablespace name of a partitioned table.
schema	character varying(64)	Name of a namespace.
subpartition_count	bigint	Number of level-2 partitions.
high_value_length	integer	Length of the partition boundary value expression.
composite	character varying(3)	Specifies whether the table is a level-2 partitioned table.
partition_position	numeric	Position of the partition in the table.
pct_free	numeric	Minimum percentage of available space in a block.
pct_used	numeric	Not supported. Its value is **NULL**.
ini_trans	numeric	Initial number of transactions. The default value is **4**. The value is **NULL** for non-Ustore partitioned tables.

Name	Type	Description
max_trans	numeric	Maximum number of transactions. The default value is **128**. The value is **NULL** for non-Ustore partitioned tables.
initial_extent	numeric	Not supported. Its value is **NULL**.
next_extent	numeric	Not supported. Its value is **NULL**.
min_extent	numeric	Not supported. Its value is **NULL**.
max_extent	numeric	Not supported. Its value is **NULL**.
max_size	numeric	Not supported. Its value is **NULL**.
pct_increase	numeric	Not supported. Its value is **NULL**.
freelists	numeric	Not supported. Its value is **NULL**.
freelist_groups	numeric	Not supported. Its value is **NULL**.
logging	character varying(7)	Specifies whether changes to a table are logged.
compression	character varying(8)	Actual compression attribute of a partitioned table.
compress_for	character varying(30)	Not supported. Its value is **NULL**.
num_rows	numeric	Number of rows in a partition.
blocks	numeric	Not supported. Its value is **NULL**.
empty_blocks	numeric	Not supported. Its value is **NULL**.
avg_space	numeric	Not supported. Its value is **NULL**.
chain_cnt	numeric	Not supported. Its value is **NULL**.
avg_row_len	numeric	Not supported. Its value is **NULL**.
sample_size	numeric	Not supported. Its value is **NULL**.
last_analyzed	timestamp with time zone	Last date when the partition was analyzed.
buffer_pool	character varying(7)	Buffer pool allocated to a partitioned block.
flash_cache	character varying(7)	Not supported. Its value is **NULL**.
cell_flash_cache	character varying(7)	Not supported. Its value is **NULL**.
global_stats	character varying(3)	Not supported. Its value is **NULL**.
user_stats	character varying(3)	Not supported. Its value is **NULL**.

Name	Type	Description
is_nested	character varying(3)	Specifies whether this partitioned table is a nested partitioned table.
parent_table_partition	character varying(128)	Not supported. Its value is **NULL**.
interval	character varying(3)	Specifies whether the partition is a partition in an interval partitioned table.
segment_created	character varying(4)	Specifies whether a table partition has segments created.
indexing	character varying(4)	Not supported. Its value is **NULL**.
read_only	character varying(4)	Not supported. Its value is **NULL**.
inmemory	character varying(8)	Not supported. Its value is **NULL**.
inmemory_priority	character varying(8)	Not supported. Its value is **NULL**.
inmemory_distribute	character varying(15)	Not supported. Its value is **NULL**.
inmemory_compression	character varying(17)	Not supported. Its value is **NULL**.
inmemory_duplicate	character varying(13)	Not supported. Its value is **NULL**.
cellmemory	character varying(24)	Not supported. Its value is **NULL**.
inmemory_service	character varying(12)	Not supported. Its value is **NULL**.
inmemory_service_name	character varying(100)	Not supported. Its value is **NULL**.
memoptimize_read	character varying(8)	Not supported. Its value is **NULL**.
memoptimize_write	character varying(8)	Not supported. Its value is **NULL**.

12.3.1.19 DB_TAB_SUBPARTITIONS

DB_TAB_SUBPARTITIONS displays information about level-2 partitioned tables accessible to the current user. It is accessible to all users. This view exists in both PG_CATALOG and SYS schema.

Table 12-144 DB_TAB_SUBPARTITIONS columns

Name	Type	Description
table_owner	character varying(64)	Table owner.
table_name	character varying(64)	Relational table name.
partition_name	character varying(64)	Partition name.
subpartition_name	character varying(64)	Name of a level-2 partition.
high_value	text	Limit of a level-2 partition. • For range partitioning and interval partitioning, the upper limit of each partition is displayed. • For list partitioning, the value list of each partition is displayed. • For hash partitioning, the number of each partition is displayed.
tablespace_name	name	Tablespace name of a level-2 partitioned table.
schema	character varying(64)	Name of a namespace.
high_value_length	integer	Character length of the limit of a level-2 partition.
partition_position	numeric	Position of the partition in the table.
subpartition_position	numeric	Position of the level-2 partition in the partition.
pct_free	numeric	Minimum percentage of available space in a block.
pct_used	numeric	Not supported. Its value is **NULL**.
ini_trans	numeric	Initial number of transactions. The default value is **4**. The value is **NULL** for non-Ustore partitioned tables.
max_trans	numeric	Maximum number of transactions. The default value is **128**. The value is **NULL** for non-Ustore partitioned tables.
initial_extent	numeric	Not supported. Its value is **NULL**.
next_extent	numeric	Not supported. Its value is **NULL**.

Name	Type	Description
min_extent	numeric	Not supported. Its value is **NULL**.
max_extent	numeric	Not supported. Its value is **NULL**.
max_size	numeric	Not supported. Its value is **NULL**.
pct_increase	numeric	Not supported. Its value is **NULL**.
freelists	numeric	Not supported. Its value is **NULL**.
freelist_groups	numeric	Not supported. Its value is **NULL**.
logging	character varying(3)	Specifies whether changes to a table are logged.
compression	character varying(8)	Actual compression attribute of a partitioned table.
compress_for	character varying(30)	Not supported. Its value is **NULL**.
num_rows	numeric	Number of rows in a level-2 partition.
blocks	numeric	Not supported. Its value is **NULL**.
empty_blocks	numeric	Not supported. Its value is **NULL**.
avg_space	numeric	Not supported. Its value is **NULL**.
chain_cnt	numeric	Not supported. Its value is **NULL**.
avg_row_len	numeric	Not supported. Its value is **NULL**.
sample_size	numeric	Not supported. Its value is **NULL**.
last_analyzed	timestamp with time zone	Last date when the partition was analyzed.
buffer_pool	character varying(7)	Buffer pool of a level-2 partition.
flash_cache	character varying(7)	Not supported. Its value is **NULL**.
cell_flash_cache	character varying(7)	Not supported. Its value is **NULL**.
global_stats	character varying(3)	Not supported. Its value is **NULL**.
user_stats	character varying(3)	Not supported. Its value is **NULL**.
interval	character varying(3)	Specifies whether the partition is a partition in an interval partitioned table.
segment_created	character varying(3)	Specifies whether a segment is created in a level-2 partitioned table.
indexing	character varying(3)	Not supported. Its value is **NULL**.

Name	Type	Description
read_only	character varying(3)	Not supported. Its value is **NULL**.
inmemory	character varying(8)	Not supported. Its value is **NULL**.
inmemory_priority	character varying(8)	Not supported. Its value is **NULL**.
inmemory_distribute	character varying(15)	Not supported. Its value is **NULL**.
inmemory_compression	character varying(17)	Not supported. Its value is **NULL**.
inmemory_duplicate	character varying(13)	Not supported. Its value is **NULL**.
inmemory_service	character varying(12)	Not supported. Its value is **NULL**.
inmemory_service_name	character varying(1000)	Not supported. Its value is **NULL**.
cellmemory	character varying(24)	Not supported. Its value is **NULL**.
memoptimize_read	character varying(8)	Not supported. Its value is **NULL**.
memoptimize_write	character varying(8)	Not supported. Its value is **NULL**.

12.3.1.20 MY_IND_PARTITIONS

MY_IND_PARTITIONS displays the partition information about local indexes of level-1 partitioned table accessible to the current user (excluding global indexes of partitioned tables). It is accessible to all users. This view exists in both PG_CATALOG and SYS schema.

Table 12-145 MY_IND_PARTITIONS columns

Name	Type	Description
index_owner	character varying(64)	Name of the owner of the partitioned table index to which an index partition belongs.
index_name	character varying(64)	Name of the partitioned table index to which the index partition belongs.
partition_name	character varying(64)	Name of an index partition.

Name	Type	Description
def_tablespace_name	name	Tablespace name of the index partition.
high_value	text	Upper limit of the partition corresponding to the index partition.
index_partition_usable	Boolean	Specifies whether an index partition is available: • **t** (true): yes. • **f** (false): no.
schema	character varying(64)	Schema of the partitioned table index to which an index partition belongs.
high_value_length	integer	Character length of the limit of the partition corresponding to an index partition.
composite	character varying(3)	Specifies whether the index is a local index on the level-2 partitioned table. This table does not store level-2 partition information. Therefore, the value is **NO**.
subpartition_count	numeric	Number of level-2 partitions in a partition. This table does not store level-2 partition information. Therefore, the value is **0**.
partition_position	numeric	Position of an index partition in the index.
status	character varying(8)	Specifies whether an index partition is available.
tablespace_name	name	Name of the tablespace that contains the partition.
pct_free	numeric	Percentage of minimum available space in a block.

Name	Type	Description
ini_trans	numeric	Initial number of transactions. The default value is **4**. The value is **NULL** for non-Ustore partitioned tables.
max_trans	numeric	Maximum number of transactions. The default value is **128**. The value is **NULL** for non-Ustore partitioned tables.
initial_extent	numeric	Not supported. Its value is **NULL**.
next_extent	numeric	Not supported. Its value is **NULL**.
min_extent	numeric	Not supported. Its value is **NULL**.
max_extent	numeric	Not supported. Its value is **NULL**.
max_size	numeric	Not supported. Its value is **NULL**.
pct_increase	numeric	Not supported. Its value is **NULL**.
freelists	numeric	Not supported. Its value is **NULL**.
freelist_groups	numeric	Not supported. Its value is **NULL**.
logging	character varying(7)	Specifies whether changes to an index are logged.
compression	character varying(13)	Specifies whether an index compression is enabled for a partitioned index.
blevel	numeric	Not supported. Its value is **NULL**.
leaf_blocks	numeric	Not supported. Its value is **NULL**.
distinct_keys	numeric	Not supported. Its value is **NULL**.

Name	Type	Description
avg_leaf_blocks_per_key	numeric	Not supported. Its value is **NULL**.
avg_data_blocks_per_key	numeric	Not supported. Its value is **NULL**.
clustering_factor	numeric	Sequence of a row in the table based on the value of the index. You need to run the **analyze** command to collect statistics.
num_rows	numeric	Number of rows in a partition. You need to run the vacuum command to collect statistics.
sample_size	numeric	Not supported. Its value is **NULL**.
last_analyzed	timestamp with time zone	Last date when the partition was analyzed. Database restart is not supported. Otherwise, data loss will occur.
buffer_pool	character varying(7)	Actual buffer pool of a partition.
flash_cache	character varying(7)	Not supported. Its value is **NULL**.
cell_flash_cache	character varying(7)	Not supported. Its value is **NULL**.
user_stats	character varying(3)	Not supported. Its value is **NULL**.
pct_direct_access	numeric	Not supported. Its value is **NULL**.
global_stats	character varying(3)	Not supported. Its value is **NULL**.
domidx_opstatus	character varying(6)	Not supported. Its value is **NULL**.
parameters	character varying(1000)	Not supported. Its value is **NULL**.

Name	Type	Description
interval	character varying(3)	Specifies whether the partition is a partition in an interval partitioned table.
segment_created	character varying(3)	Specifies whether an index partition segment has been created.
orphaned_entries	character varying(3)	Not supported. Its value is **NULL**.

12.3.1.21 MY_IND_SUBPARTITIONS

MY_IND_SUBPARTITIONS displays the partition information about local indexes of level-2 partitioned table owned by the current user (excluding global indexes of partitioned tables). It is accessible to all users. This view exists in the PG_CATALOG and SYS schemas.

Table 12-146 MY_IND_SUBPARTITIONS columns

Name	Type	Description
index_owner	character varying(64)	Owner name of the partitioned table index to which an index partition belongs.
index_name	character varying(64)	Index name of the partitioned table index to which an index partition belongs.
partition_name	character varying(64)	Name of the level-1 partition where an index is located.
subpartition_name	character varying(64)	Name of the level-2 partition where an index is located.
def_tablespace_name	name	Tablespace name of an index partition.
high_value	text	Limit of the partition corresponding to an index partition.

Name	Type	Description
index_partition_usable	Boolean	Specifies whether an index partition is available: • **t** (true): yes • **f** (false): no
schema	character varying(64)	Schema of the partitioned table index to which an index partition belongs.
high_value_length	integer	Character length of the limit of the partition corresponding to an index partition.
partition_position	numeric	Position of an index partition in the index.
subpartition_position	numeric	Location of the level-2 partition in the partition.
status	character varying(8)	Specifies whether the index partition is available.
tablespace_name	name	Tablespace name of an index partition.
pct_free	numeric	Minimum percentage of available space in a block.
ini_trans	numeric	Initial number of transactions. The default value is **4**. The value is **NULL** for non-Ustore partitioned tables.
max_trans	numeric	Maximum number of transactions. The default value is **128**. The value is **NULL** for non-Ustore partitioned tables.
initial_extent	numeric	Not supported. The value is **NULL**.
next_extent	numeric	Not supported. The value is **NULL**.
min_extent	numeric	Not supported. The value is **NULL**.

Name	Type	Description
max_extent	numeric	Not supported. The value is **NULL**.
max_size	numeric	Not supported. The value is **NULL**.
pct_increase	numeric	Not supported. The value is **NULL**.
freelists	numeric	Not supported. The value is **NULL**.
freelist_groups	numeric	Not supported. The value is **NULL**.
logging	character varying(7)	Specifies whether changes to an index are recorded.
compression	character varying(13)	Compression type used for level-2 partitions.
blevel	numeric	Not supported. The value is **NULL**.
leaf_blocks	numeric	Not supported. The value is **NULL**.
distinct_keys	numeric	Not supported. The value is **NULL**.
avg_leaf_blocks_per_key	numeric	Not supported. The value is **NULL**.
avg_data_blocks_per_key	numeric	Not supported. The value is **NULL**.
clustering_factor	numeric	Sequence of a row in the table based on the value of the index. You need to run the analyze command to collect statistics.
num_rows	numeric	Number of rows in a level-2 partition. You need to run the vacuum command to collect statistics.
sample_size	numeric	Not supported. The value is **NULL**.

Name	Type	Description
last_analyzed	timestamp with time zone	Last date when the partition was analyzed. Database restart is not supported. Otherwise, data loss will occur.
buffer_pool	character varying(7)	Buffer pool of a level-2 partition.
flash_cache	character varying(7)	Not supported. The value is **NULL**.
cell_flash_cache	character varying(7)	Not supported. The value is **NULL**.
user_stats	character varying(3)	Not supported. The value is **NULL**.
global_stats	character varying(3)	Not supported. The value is **NULL**.
interval	character varying(3)	Specifies whether the partition is a partition in an interval partitioned table.
segment_created	character varying(3)	Specifies whether an index partition segment has been created.
domidx_opstatus	character varying(6)	Not supported. The value is **NULL**.
parameters	character varying(1000)	Not supported. The value is **NULL**.

12.3.1.22 MY_PART_COL_STATISTICS

MY_PART_COL_STATISTICS displays column statistics and histogram information about table partitions owned by the current user. It is accessible to all users. This view exists in the PG_CATALOG and SYS schemas.

Table 12-147 MY_PART_COL_STATISTICS columns

Name	Type	Description
table_name	character varying(128)	Table name.
partition_name	character varying(128)	Table partition name
column_name	character varying(4000)	Column name.

Name	Type	Description
num_distinct	numeric	Not supported. Set it to **NULL**.
low_value	raw	Not supported. Set it to **NULL**.
high_value	raw	Not supported. Set it to **NULL**.
density	numeric	Not supported. Set it to **NULL**.
num_nulls	numeric	Not supported. Set it to **NULL**.
num_buckets	numeric	Not supported. Set it to **NULL**.
sample_size	numeric	Not supported. Set it to **NULL**.
last_analyzed	timestamp(0) without time zone	Not supported. Set it to **NULL**.
global_stats	character varying(3)	Not supported. Set it to **NULL**.
user_stats	character varying(3)	Not supported. Set it to **NULL**.
notes	character varying(63)	Not supported. Set it to **NULL**.
avg_col_len	numeric	Not supported. Set it to **NULL**.
histogram	character varying(15)	Not supported. Set it to **NULL**.
schema	character varying(64)	Name of the namespace to which the column belongs.

12.3.1.23 MY_PART_INDEXES

MY_PART_INDEXES displays information about partitioned table indexes accessible to the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-148 MY_PART_INDEXES columns

Name	Type	Description
def_tablespace_name	name	Tablespace name of a partitioned table index

Name	Type	Description
index_owner	character varying(64)	Owner name of a partitioned table index
index_name	character varying(64)	Name of a partitioned table index
partition_count	bigint	Number of index partitions of a partitioned table index
partitioning_key_count	integer	Number of partition keys of a partitioned table
partitioning_type	text	Partitioning policy of a partitioned table **NOTE** For details about the supported partitioning policies of the current partitioned table, see **CREATE TABLE PARTITION**.
schema	character varying(64)	Schema of a partitioned table index
table_name	character varying(64)	Name of the partitioned table to which a partitioned table index belongs
subpartitioning_type	text	Partitioning policy of a level-2 partitioned table. If the partitioned table is a level-1 partitioned table, **NONE** is displayed. **NOTE** For details about the supported partitioning policies of the current level-2 partitioned table, see **CREATE TABLE SUBPARTITION**.
def_subpartition_count	integer	Default number of level-2 partitions to be created. The value is **1** for a level-2 partitioned table and **0** for a level-1 partitioned table.
subpartitioning_key_count	integer	Number of level-2 partition keys of the partitioned table.

12.3.1.24 MY_PART_KEY_COLUMNS

MY_PART_KEY_COLUMNS displays information about the partition key columns of partitioned tables or partitioned indexes owned by the current user. This view can be accessed by all users. Only the information about the current user can be viewed. This view exists in both PG_CATALOG and SYS schema.

Table 12-149 MY_PART_KEY_COLUMNS columns

Name	Type	Description
name	character varying(128)	Name of a partitioned table or index.
object_type	character(5)	Object type. • Its value is **table** for a partitioned table. • Its value is **index** for a partitioned index.
column_name	character varying(4000)	Partition key column name of a partitioned table or index.
column_position	numeric	Position of a column in a partition.
collated_column_id	numeric	Not supported. Its value is **NULL**.

12.3.1.25 MY_PART_TABLES

MY_PART_TABLES displays information about partitioned tables accessible to the current user. This view exists in both PG_CATALOG and SYS schema.

Table 12-150 MY_PART_TABLES columns

Name	Type	Description
table_owner	character varying(64)	Owner name of a partitioned table.
table_name	character varying(64)	Name of a partitioned table.
partitioning_type	text	Partitioning policy of a partitioned table. **NOTE** For details about the supported partitioning policies of the current partitioned table, see **CREATE TABLE PARTITION**.
partition_count	bigint	Number of partitions of a partitioned table.
partitioning_key_count	integer	Number of partition keys of a partitioned table.
def_tablespace_name	name	Tablespace name of a partitioned table.
schema	character varying(64)	Schema of a partitioned table.

Name	Type	Description
subpartitioning_type	text	Partitioning policy of a level-2 partitioned table. If the partitioned table is a level-1 partitioned table, **NONE** is displayed. **NOTE** For details about the supported partitioning policies of the current level-2 partitioned table, see **CREATE TABLE SUBPARTITION**.
def_subpartition_count	integer	Default number of level-2 partitions to be created. The value is **1** for a level-2 partitioned table and **0** for a level-1 partitioned table.
subpartitioning_key_count	integer	Number of level-2 partition keys of the partitioned table.
status	character varying(8)	Not supported. The value is **valid**.
def_pct_free	numeric	Default value of **PCTFREE** used when a partition is added.
def_pct_used	numeric	Not supported. Its value is **NULL**.
def_ini_trans	numeric	Default value of **INITRANS** used when a partition is added.
def_max_trans	numeric	Default value of **MAXTRANS** used when a partition is added.
def_initial_extent	character varying(40)	Not supported. Its value is **NULL**.
def_next_extent	character varying(40)	Not supported. Its value is **NULL**.
def_min_extents	character varying(40)	Not supported. Its value is **NULL**.
def_max_extents	character varying(40)	Not supported. Its value is **NULL**.
def_max_size	character varying(40)	Not supported. Its value is **NULL**.

Name	Type	Description
def_pct_increase	character varying(40)	Not supported. Its value is **NULL**.
def_freelists	numeric	Not supported. Its value is **NULL**.
def_freelist_groups	numeric	Not supported. Its value is **NULL**.
def_logging	character varying(7)	Not supported. Its value is **NULL**.
def_compression	character varying(8)	Default compression mode used when a partition is added. The options are as follows: • **NONE** • **ENABLED** • **DISABLED**
def_compress_for	character varying(30)	Default compression mode used when a partition is added. **NOTE** For available compression modes and compression levels, see **WITH ({ storage_parameter = value } [, ...])**.
def_buffer_pool	character varying(7)	Not supported. The value is **DEFAULT**.
def_flash_cache	character varying(7)	Not supported. Its value is **NULL**.
def_cell_flash_cache	character varying(7)	Not supported. Its value is **NULL**.
ref_ptn_constraint_name	character varying(128)	Not supported. Its value is **NULL**.
interval	character varying(1000)	Interval.
autolist	text	Specifies whether to enable the automatic extension function for the level-1 list partition. • **YES**: Enable the automatic extension function. • **NO**: Disable the automatic extension function.

Name	Type	Description
interval_subpartition	character varying(1000)	Not supported. Its value is **NULL**.
autolist_subpartition	text	Specifies whether to enable the automatic extension function for the level-2 list partition. • **YES**: Enable the automatic extension function. • **NO**: Disable the automatic extension function.
is_nested	character varying(3)	Not supported. The value is **NO**.
def_segment_creation	character varying(4)	Currently, the segment-page mode is not supported. When the segment-page mode is enabled, the value is **YES**.
def_indexing	character varying(3)	Not supported. The value is **ON**.
def_inmemory	character varying(8)	Not supported. The value is **NONE**.
def_inmemory_priority	character varying(8)	Not supported. Its value is **NULL**.
def_inmemory_distribute	character varying(15)	Not supported. Its value is **NULL**.
def_inmemory_compression	character varying(17)	Not supported. Its value is **NULL**.
def_inmemory_duplicate	character varying(13)	Not supported. Its value is **NULL**.
def_read_only	character varying(3)	Not supported. The value is **NO**.
def_cellmemory	character varying(24)	Not supported. Its value is **NULL**.
def_inmemory_service	character varying(12)	Not supported. Its value is **NULL**.
def_inmemory_service_name	character varying(1000)	Not supported. Its value is **NULL**.

12.3.1.26 MY_SUBPART_COL_STATISTICS

MY_SUBPART_COL_STATISTICS displays column statistics and histogram information about level-2 partitions of partitioned objects owned by the current

user. All users can access this view. This view exists in the PG_CATALOG and SYS schemas.

Table 12-151 MY_SUBPART_COL_STATISTICS columns

Name	Type	Description
table_name	character varying(128)	Table name.
subpartition_name	character varying(128)	Name of a level-2 partition.
column_name	character varying(4000)	Column name.
num_distinct	numeric	Not supported. Set it to **NULL**.
low_value	raw	Not supported. Set it to **NULL**.
high_value	raw	Not supported. Set it to **NULL**.
density	numeric	Not supported. Set it to **NULL**.
num_nulls	numeric	Not supported. Set it to **NULL**.
num_buckets	numeric	Not supported. Set it to **NULL**.
sample_size	numeric	Not supported. Set it to **NULL**.
last_analyzed	timestamp(0) without time zone	Not supported. Set it to **NULL**.
global_stats	character varying(3)	Not supported. Set it to **NULL**.
user_stats	character varying(3)	Not supported. Set it to **NULL**.
notes	character varying(41)	Not supported. Set it to **NULL**.
avg_col_len	numeric	Not supported. Set it to **NULL**.
histogram	character varying(15)	Not supported. Set it to **NULL**.
schema	character varying(64)	Name of the namespace to which the column belongs.

12.3.1.27 MY_SUBPART_KEY_COLUMNS

MY_SUBPART_KEY_COLUMNS displays information about the partition key columns of level-2 partitioned tables or partitioned indexes owned by the current user. This view can be accessed by all users. Only the information about the current user can be viewed. This view exists in both PG_CATALOG and SYS schema.

Table 12-152 MY_SUBPART_KEY_COLUMNS columns

Name	Type	Description
name	character varying(128)	Name of a level-2 partitioned table or index.
object_type	character(5)	Object type. • Its value is **table** for a partitioned table. • Its value is **index** for a partitioned index.
column_name	character varying(4000)	Partition key column name of a level-2 partitioned table or index.
column_position	numeric	Position of a column in a partition.
collated_column_id	numeric	Not supported. Its value is **NULL**.

12.3.1.28 MY_TAB_PARTITIONS

MY_TAB_PARTITIONS displays all level-1 partitions accessible to the current user. Each level-1 table partition of a partitioned table accessible to the current user has one record in MY_TAB_PARTITIONS. It is accessible to all users. This view exists in both PG_CATALOG and SYS schema.

Table 12-153 MY_TAB_PARTITIONS columns

Name	Type	Description
table_owner	character varying(64)	Table owner.
table_name	character varying(64)	Relational table name.
partition_name	character varying(64)	Partition name.

Name	Type	Description
high_value	text	Limit of a partition. • For range partitioning and interval partitioning, the upper limit of each partition is displayed. • For list partitioning, the value list of each partition is displayed. • For hash partitioning, the number of each partition is displayed.
tablespace_name	name	Tablespace name of a partitioned table.
schema	character varying(64)	Name of a namespace.
subpartition_count	bigint	Number of level-2 partitions.
high_value_length	integer	Length of the partition boundary value expression.
composite	character varying(3)	Specifies whether the table is a level-2 partitioned table.
partition_position	numeric	Position of the partition in the table.
pct_free	numeric	Minimum percentage of available space in a block.
pct_used	numeric	Not supported. Its value is **NULL**.
ini_trans	numeric	Initial number of transactions. The default value is **4**. The value is **NULL** for non-Ustore partitioned tables.
max_trans	numeric	Maximum number of transactions. The default value is **128**. The value is **NULL** for non-Ustore partitioned tables.
initial_extent	numeric	Not supported. Its value is **NULL**.
next_extent	numeric	Not supported. Its value is **NULL**.
min_extent	numeric	Not supported. Its value is **NULL**.
max_extent	numeric	Not supported. Its value is **NULL**.
max_size	numeric	Not supported. Its value is **NULL**.

Name	Type	Description
pct_increase	numeric	Not supported. Its value is **NULL**.
freelists	numeric	Not supported. Its value is **NULL**.
freelist_groups	numeric	Not supported. Its value is **NULL**.
logging	character varying(7)	Specifies whether changes to a table are logged.
compression	character varying(8)	Actual compression attribute of a partitioned table.
compress_for	character varying(30)	Not supported. Its value is **NULL**.
num_rows	numeric	Number of rows in a partition.
blocks	numeric	Not supported. Its value is **NULL**.
empty_blocks	numeric	Not supported. Its value is **NULL**.
avg_space	numeric	Not supported. Its value is **NULL**.
chain_cnt	numeric	Not supported. Its value is **NULL**.
avg_row_len	numeric	Not supported. Its value is **NULL**.
sample_size	numeric	Not supported. Its value is **NULL**.
last_analyzed	timestamp with time zone	Last date when the partition was analyzed.
buffer_pool	character varying(7)	Buffer pool allocated to a partitioned block.
flash_cache	character varying(7)	Not supported. Its value is **NULL**.
cell_flash_cache	character varying(7)	Not supported. Its value is **NULL**.
global_stats	character varying(3)	Not supported. Its value is **NULL**.
user_stats	character varying(3)	Not supported. Its value is **NULL**.
is_nested	character varying(3)	Specifies whether this partitioned table is a nested partitioned table.
parent_table_partition	character varying(128)	Not supported. Its value is **NULL**.
interval	character varying(3)	Specifies whether the partition is a partition in an interval partitioned table.
segment_created	character varying(4)	Specifies whether a table partition has segments created.
indexing	character varying(4)	Not supported. Its value is **NULL**.

Name	Type	Description
read_only	character varying(4)	Not supported. Its value is **NULL**.
inmemory	character varying(8)	Not supported. Its value is **NULL**.
inmemory_priority	character varying(8)	Not supported. Its value is **NULL**.
inmemory_distribute	character varying(15)	Not supported. Its value is **NULL**.
inmemory_compression	character varying(17)	Not supported. Its value is **NULL**.
inmemory_duplicate	character varying(13)	Not supported. Its value is **NULL**.
cellmemory	character varying(24)	Not supported. Its value is **NULL**.
inmemory_service	character varying(12)	Not supported. Its value is **NULL**.
inmemory_service_name	character varying(100)	Not supported. Its value is **NULL**.
memoptimize_read	character varying(8)	Not supported. Its value is **NULL**.
memoptimize_write	character varying(8)	Not supported. Its value is **NULL**.

12.3.1.29 MY_TAB_SUBPARTITIONS

MY_TAB_SUBPARTITIONS displays information about all level-2 partitioned tables accessible to the current user. This view can be accessed by all users. Only the information about the current user can be viewed. This view exists in both PG_CATALOG and SYS schema.

Table 12-154 MY_TAB_SUBPARTITIONS columns

Name	Type	Description
table_owner	character varying(64)	Table owner.
schema	character varying(64)	Schema name.
table_name	character varying(64)	Table name.
partition_name	character varying(64)	Partition name.

Name	Type	Description
subpartition_name	character varying(64)	Name of a level-2 partition.
high_value	text	Boundary value expression in a level-2 partition.
high_value_length	integer	Length of the boundary value expression in level-2 partition.
partition_position	numeric	Position of the partition in the table.
subpartition_position	numeric	Position of the level-2 partition in the partition.
tablespace_name	name	Name of the tablespace where the level-2 partition is located.
pct_free	numeric	Percentage of minimum available space in a block.
pct_used	numeric	Not supported. Its value is **NULL**.
ini_trans	numeric	Initial number of transactions.
max_trans	numeric	Maximum number of transactions.
initial_extent	numeric	Not supported. Its value is **NULL**.
next_extent	numeric	Not supported. Its value is **NULL**.
min_extent	numeric	Not supported. Its value is **NULL**.
max_extent	numeric	Not supported. Its value is **NULL**.
max_size	numeric	Not supported. Its value is **NULL**.
pct_increase	numeric	Not supported. Its value is **NULL**.
freelists	numeric	Not supported. Its value is **NULL**.
freelist_groups	numeric	Not supported. Its value is **NULL**.
logging	character varying(7)	Specifies whether changes to a table are logged. • **YES** • **NO**
compression	character varying(8)	Specifies whether a subpartition is compressed. Value range: **ENABLED** and **DISABLED**.
compress_for	character varying(30)	Not supported. Its value is **NULL**.
num_rows	numeric	Not supported. Its value is **NULL**.
blocks	numeric	Not supported. Its value is **NULL**.

Name	Type	Description
empty_blocks	numeric	Not supported. Its value is **NULL**.
avg_space	numeric	Not supported. Its value is **NULL**.
chain_cnt	numeric	Not supported. Its value is **NULL**.
avg_row_len	numeric	Not supported. Its value is **NULL**.
sample_size	numeric	Not supported. Its value is **NULL**.
last_analyzed	timestamp with time zone	Date when the table was last analyzed. Database restart is not supported. Otherwise, data loss will occur.
buffer_pool	character varying(7)	Buffer pool of a level-2 partition. • **DEFAULT** • **KEEP** • **RECYCLE** • **NULL**
flash_cache	character varying(7)	Not supported. Its value is **NULL**.
cell_flash_cache	character varying(7)	Not supported. Its value is **NULL**.
global_stats	character varying(3)	Not supported. Its value is **NULL**.
user_stats	character varying(3)	Not supported. Its value is **NULL**.
interval	character varying(3)	Specifies whether the partition is in the interval section of an interval partitioned table (**YES**) or whether the partition is in the range section (**NO**).
segment_created	character varying(3)	Specifies whether the level-2 partition segment of the table has been created. • **YES**: created. • **NO**: not created. • **N/A**: The table does not have level-2 partitions.
indexing	character varying(3)	Not supported. Its value is **NULL**.
read_only	character varying(5)	Not supported. Its value is **NULL**.
inmemory	character varying(8)	Not supported. Its value is **NULL**.

Name	Type	Description
inmemory_priority	character varying(8)	Not supported. Its value is **NULL**.
inmemory_distribute	character varying(15)	Not supported. Its value is **NULL**.
inmemory_compression	character varying(17)	Not supported. Its value is **NULL**.
inmemory_duplicate	character varying(13)	Not supported. Its value is **NULL**.
inmemory_service	character varying(12)	Not supported. Its value is **NULL**.
inmemory_service_name	character varying(1000)	Not supported. Its value is **NULL**.
cellmemory	character varying(24)	Not supported. Its value is **NULL**.
memoptimize_read	character varying(8)	Not supported. Its value is **NULL**.
memoptimize_write	character varying(8)	Not supported. Its value is **NULL**.

12.3.1.30 GS_STATIO_ALL_PARTITIONS

GS_STATIO_ALL_PARTITIONS contains I/O statistics about each partition in a partitioned table of the current database. The information can be queried by using the gs_statio_get_all_partitions_stats() function.

Table 12-155 GS_STATIO_ALL_PARTITIONS columns

Name	Type	Description
partition_oid	oid	Specifies a partition OID.
schemaname	name	Specifies the name of a partition mode.
relname	name	Specifies the name of a table to which a partition belongs.
partition_name	name	Specifies the name of a level-1 partition to which a partition belongs.
sub_partition_name	name	Specifies the name of a level-2 partition to which a partition belongs.
heap_blks_read	bigint	Specifies the number of disk blocks read from a partition.

Name	Type	Description
heap_blks_hit	bigint	Specifies the number of cache hits in a partition.
idx_blks_read	bigint	Specifies the number of disk blocks read from all indexes in a partition.
idx_blks_hit	bigint	Specifies the number of cache hits of all indexes in a partition.
toast_blks_read	bigint	Specifies the number of disk blocks read from TOAST table partitions (if any) in a partition.
toast_blks_hit	bigint	Specifies the number of buffer hits in TOAST tables (if any) in a partition.
tidx_blks_read	bigint	Specifies the number of disk blocks read from TOAST table partitioned indexes (if any) in a partition.
tidx_blks_hit	bigint	Specifies the number of buffer hits in TOAST table indexes (if any) in a partition.

12.3.1.31 GS_STAT_XACT_ALL_PARTITIONS

GS_STAT_XACT_ALL_PARTITIONS displays the transaction status about all partitions of partitioned tables in a namespace. The information can be queried by using the gs_stat_get_xact_all_partitions_stats() function.

Table 12-156 GS_STAT_XACT_ALL_PARTITIONS columns

Name	Type	Description
partition_oid	oid	Partition OID.
schemaname	name	Name of a partition schema.
relname	name	Name of a table where the partition is located.
partition_name	name	Name of a level-1 partition to which the partition belongs.
sub_partition_name	name	Name of the level-2 partition to which the partition belongs.
seq_scan	bigint	Number of sequential scans initiated by a partition.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated by a partition.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.

Name	Type	Description
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)

12.3.1.32 GS_STAT_ALL_PARTITIONS

GS_STAT_ALL_PARTITIONS displays information about each partition in all partitioned tables in the current database. Each partition occupies one row (only its level-2 partitions are displayed for level-2 partitioned tables). The view information is queried by the gs_stat_get_all_partitions_stats() function.

Table 12-157 GS_STAT_ALL_PARTITIONS columns

Name	Type	Description
partition_oid	oid	Partition OID.
schemaname	name	Schema name of a table to which the partition belongs.
relname	name	Name of a table where the partition is located.
partition_name	name	Name of a level-1 partition to which the partition belongs.
sub_partition_name	name	Name of the level-2 partition to which the partition belongs.
seq_scan	bigint	Number of sequential scans initiated by a partition.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated by a partition.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.

Name	Type	Description
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)
n_live_tup	bigint	Estimated number of live rows.
n_dead_tup	bigint	Estimated number of inactive rows.
last_vacuum	timestamp with time zone	Time when the partition was last cleared.
last_autovacuum	timestamp with time zone	Time when the partition was last cleared by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Time when the partition was last analyzed.
last_autoanalyze	timestamp with time zone	Time when the partition was last analyzed by the autovacuum daemon thread.
vacuum_count	bigint	Number of times that a partition is cleared.
autovacuum_count	bigint	Number of times that a partition is cleared by the autovacuum daemon thread.
analyze_count	bigint	Number of times that a partition is analyzed.
autoanalyze_count	bigint	Number of times that a partition is analyzed by the autovacuum daemon thread.

12.3.2 OLTP Table Compression

12.3.2.1 GS_ADM_ILMDATAMOVEMENTPOLICIES

GS_ADM_ILMDATAMOVEMENTPOLICIES displays the data movement summary of ILM policies, including the policy name, action type, and conditions. Only the system administrator can access this system view.

Table 12-158 GS_ADM_ILMDATAMOVEMENTPOLICIES columns

Name	Type	Description
policy_name	character varying(128)	Name of an ADO policy, which is automatically generated. The name format is P + Policy ID.

Name	Type	Description
action_type	character varying(11)	Action type. The current version supports only compression.
scope	character varying(7)	Scope. In the current version, only rows are supported.
compression_level	character varying(30)	Compression level. This parameter is available when the action type is compression.
tier_tablespace	character varying(128)	Target space. This parameter is available only when the action type is migration. The value is **null** in the current version.
tier_status	character varying(9)	Specifies whether the target space is read-only. The value is **null** in the current version.
condition_type	character varying(22)	Condition type. The current version supports only the last modification time.
condition_days	numeric	Number of days as a condition.
custom_function	character varying(128)	Name of a user-defined function. The value is **null** in the current version.
policy_subtype	character varying(10)	Policy subtype. The value is **null** in the current version.
action_clause	clob	Text that can be automatically executed during policy execution. The value is **null** in the current version.
tier_to	character varying(10)	Type of the target database to which data is migrated. The value is **null** in the current version.

12.3.2.2 GS_ADM_ILMOBJECTS

GS_ADM_ILMOBJECTS displays the brief information about all data objects to which ILM policies are applied and the corresponding policies, including the policy name, data object name, policy source, and policy enabling/disabling status. Only the system administrator can access this system view.

Table 12-159 GS_ADM_ILMOBJECTS columns

Name	Type	Description
policy_name	character varying(128)	Name of an ADO policy, which is automatically generated. The name format is P + Policy ID.

Name	Type	Description
object_owner	character varying(128)	Name of the schema where the data object is located.
object_name	character varying(128)	Data object name.
subobject_name	character varying(128)	Data object partition name or level-2 partition name.
object_type	character varying(18)	Data object type. • **r**: table • **p**: partition • **s**: level-2 partition
inherited_from	character varying(20)	Object whose policy is inherited by the current policy. The options are as follows: • **TABLE**: table • **TABLE PARTITION**: partition • **POLICY NOT INHERITED**: not inherited
tbs_inherited_from	character varying(30)	TS whose policy is inherited by the current policy. The value is **null** in the current version.
enabled	character varying(7)	Specifies whether the policy is enabled for the current object.
deleted	character varying(7)	Specifies that the ILM policy on the object is deleted. If the partition or the policy on the partition is deleted, the record (ILMOBJ) is deleted.

12.3.2.3 GS_ADM_ILMPOLICIES

GS_ADM_ILMPOLICIES displays the brief information about an ILM policy, including the policy name, type, enabling status, disabling status, and deletion status. Only the system administrator can access this system view.

Table 12-160 GS_ADM_ILMPOLICIES columns

Name	Type	Description
policy_name	character varying(128)	Name of an ADO policy, which is automatically generated. The name format is p+*Policy ID*.
policy_type	character varying(13)	Policy type.

tablespace	character varying(30)	Tablespace name. It has a value when the policy is created on a tablespace. The value is **null** in the current version.
enabled	character varying(6)	Specifies whether the policy is enabled.
deleted	character varying(7)	Specifies whether the policy is deleted.

12.3.2.4 GS_ADM_ILMEVALUATIONDETAILS

GS_ADM_ILMEVALUATIONDETAILS displays the evaluation details of an ADO task, including the task ID, policy information, object information, evaluation result, and ADO job name. By default, only the system administrator can access the system view.

Table 12-161 GS_ADM_ILMEVALUATIONDETAILS columns

Name	Type	Description
task_id	bigint	ID of an ADO task.
policy_name	character varying(128)	Name of an ADO policy, which is automatically generated. The name format is p+*Policy ID*.
object_owner	character varying(128)	Name of the schema where the data object is located.
object_name	character varying(128)	Data object name.
subobject_name	character varying(128)	Data object partition name or level-2 partition name.
object_type	character varying(18)	Data object type. • **r**: table • **p**: partition • **s**: level-2 partition
selected_for_execution	character varying(42)	Evaluation result: • **SELECTED FOR EXECUTION**: The evaluation is passed. • **PRECONDITION NOT SATISFIED**: The evaluation fails. • **JOB ALREADY EXISTS**: The task already exists.
job_name	character varying(128)	Name of the task that executes a specific ADO job. If the evaluation result is "passed", this parameter has a value.

Name	Type	Description
comments	character varying(4000)	Reserved column.

12.3.2.5 GS_ADM_ILMPARAMETERS

GS_ADM_ILMPARAMETERS displays the environment parameters related to ILM scheduling and execution. The parameters can be modified through the DBE_ILM_ADMIN.CUSTOMIZE_ILM interface. By default, only the system administrator can access the system view.

Table 12-162 GS_ADM_ILMPARAMETERS columns

Name	Type	Description
name	character varying(128)	ADO-related environment parameters, which can be modified through the DBE_ILM_ADMIN.CUSTOMIZE_ILM interface.
value	numeric	Parameter value.

12.3.2.6 GS_ADM_ILMRESULTS

GS_ADM_ILMRESULTS displays the execution details of an ADO job, including the task ID, job name, job status, and job time. By default, only the system administrator can access this system view. Common users can access the system view only after being authorized.

Table 12-163 GS_ADM_ILMRESULTS columns

Name	Type	Description
task_id	bigint	ID of an ADO task.
job_name	character varying(128)	ADO job name.
job_state	character varying(35)	ADO job status.
start_time	timestamp with time zone	Time when a job starts to be scheduled.
completion_time	timestamp with time zone	Completion time.

Name	Type	Description
comments	character varying(4000)	If a job fails, the failure cause is recorded here.
statistics	clob	Statistics.

12.3.2.7 GS_ADM_ILMTASKS

GS_ADM_ILMTASKS displays the brief information about an ADO task, including the task ID, task owner, status, and time. By default, only the system administrator can access this view. Common users can access the view only after being authorized.

Table 12-164 GS_ADM_ILMTASKS columns

Name	Type	Description
task_id	bigint	ID of an ADO task.
task_owner	character varying(128)	User who initiates an ADO task. This column exists only in the ADM view.
state	character varying(9)	Status. • **INACTIVE** • **ACTIVE** • **COMPLETED** • **UNKNOWN**: being evaluated
creation_time	timestamp with time zone	Creation time.
start_time	timestamp with time zone	Time when the status changes to active.
completion_time	timestamp with time zone	Completion time.

12.3.2.8 GS_MY_ILMEVALUATIONDETAILS

GS_MY_ILMEVALUATIONDETAILS displays the evaluation details of an ADO task, including the task ID, policy information, object information, evaluation result, and ADO job name.

Table 12-165 GS_MY_ILMEVALUATIONDETAILS columns

Name	Type	Description
task_id	bigint	ID of an ADO task.
policy_name	character varying(128)	Name of an ADO policy, which is automatically generated. The name format is P + Policy ID.
object_owner	character varying(128)	Name of the schema where the data object is located.
object_name	character varying(128)	Data object name.
subobject_name	character varying(128)	Data object partition name or level-2 partition name.
object_type	character varying(18)	Data object type. • **r**: table • **p**: partition • **s**: level-2 partition
selected_for_execution	character varying(42)	Evaluation result: • **SELECTED FOR EXECUTION**: The evaluation is passed. • **PRECONDITION NOT SATISFIED**: The evaluation fails. • **JOB ALREADY EXISTS**: The task already exists.
job_name	character varying(128)	Name of the task that executes a specific ADO job. If the evaluation result is "passed", this parameter has a value.
comments	character varying(4000)	Reserved column.

12.3.2.9 GS_MY_ILMRESULTS

GS_MY_ILMRESULTS displays the execution details of an ADO job, including the task ID, job name, job status, and job time.

Table 12-166 GS_MY_ILMRESULTS columns

Name	Type	Description
task_id	bigint	ID of an ADO task.
job_name	character varying(128)	ADO job name.

Name	Type	Description
job_state	character varying(35)	ADO job status.
start_time	timestamp with time zone	Time when a job starts to be scheduled.
completion_time	timestamp with time zone	Completion time.
comments	character varying(4000)	If a job fails, the failure cause is recorded here.
statistics	clob	Statistics.

12.3.2.10 GS_MY_ILMTASKS

GS_MY_ILMTASKS displays the brief information about an ADO task, including the task ID, task owner, status, and time.

Table 12-167 GS_MY_ILMTASKS columns

Name	Type	Description
task_id	bigint	ID of an ADO task.
task_owner	character varying(128)	User who initiates an ADO task. This column exists only in the ADM view.
state	character varying(9)	Status. • **INACTIVE** • **ACTIVE** • **COMPLETED** • **UNKNOWN**: being evaluated
creation_time	timestamp with time zone	Creation time.
start_time	timestamp with time zone	Time when the status changes to active.
completion_time	timestamp with time zone	Completion time.

12.3.2.11 GS_MY_ILMDATAMOVEMENTPOLICIES

GS_MY_ILMDATAMOVEMENTPOLICIES displays the data movement summary of ILM policies, including the policy name, action type, and conditions.

Table 12-168 GS_MY_ILMDATAMOVEMENTPOLICIES columns

Name	Type	Description
policy_name	character varying(128)	Name of an ADO policy, which is automatically generated. The name format is p + Policy ID.
action_type	character varying(11)	Action type. The current version supports only compression.
scope	character varying(7)	Scope. In the current version, only rows are supported.
compression_level	character varying(30)	Compression level. This parameter is available when the action type is compression.
tier_tablespace	character varying(128)	Target space. This parameter is available only when the action type is migration. The value is **null** in the current version.
tier_status	character varying(9)	Specifies whether the target space is read-only. The value is **null** in the current version.
condition_type	character varying(22)	Condition type. The current version supports only the last modification time.
condition_days	numeric	Number of days as a condition.
custom_function	character varying(128)	Name of a user-defined function. The value is **null** in the current version.
policy_subtype	character varying(10)	Policy subtype. The value is **null** in the current version.
action_clause	clob	Text that can be automatically executed during policy execution. The value is **null** in the current version.
tier_to	character varying(10)	Type of the target database to which data is migrated. The value is **null** in the current version.

12.3.2.12 GS_MY_ILMOBJECTS

GS_MY_ILMOBJECTS displays the brief information about all data objects to which ILM policies are applied and the corresponding policies, including the policy name, data object name, policy source, and policy enabling/disabling status.

Table 12-169 GS_MY_ILMOBJECTS columns

Name	Type	Description
policy_name	character varying(128)	Name of an ADO policy, which is automatically generated. The name format is p + Policy ID.
object_owner	character varying(128)	Name of the schema where the data object is located.
object_name	character varying(128)	Data object name.
subobject_name	character varying(128)	Data object partition name or level-2 partition name.
object_type	character varying(18)	Data object type. • **r**: table • **p**: partition • **s**: level-2 partition
inherited_from	character varying(20)	Object whose policy is inherited by the current policy. The options are as follows: • **TABLE**: table • **TABLE PARTITION**: partition • **POLICY NOT INHERITED**: not inherited
tbs_inherited_from	character varying(30)	TS whose policy is inherited by the current policy. The value is **null** in the current version.
enabled	character varying(7)	Specifies whether the policy is enabled for the current object.
deleted	character varying(7)	Specifies that the ILM policy on the object is deleted. If the partition or the policy on the partition is deleted, the record (ILMOBJ) is deleted.

12.3.2.13 GS_MY_ILMPOLICIES

GS_MY_ILMPOLICIES displays the brief information about an ILM policy, including the policy name, type, enabling status, disabling status, and deletion status.

Table 12-170 GS_MY_ILMPOLICIES columns

Name	Type	Description
policy_name	character varying(128)	Name of an ADO policy, which is automatically generated. The name format is p + Policy ID.

Name	Type	Description
policy_type	character varying(13)	Policy type.
tablespace	character varying(30)	Tablespace name. This parameter has a value when the policy is created on a tablespace. The value is **null** in the current version.
enabled	character varying(6)	Specifies whether the policy is enabled.
deleted	character varying(7)	Specifies whether the policy is deleted.

12.3.3 Communications

12.3.3.1 GS_COMM_LISTEN_ADDRESS_EXT_INFO

GS_COMM_LISTEN_ADDRESS_EXT_INFO queries DFX information, such as threads, sessions, and sockets, for connecting the extended IP addresses. Currently, this view cannot be queried.

Table 12-171 GS_COMM_LISTEN_ADDRESS_EXT_INFO columns

Name	Type	Description
node_name	text	Name of the current instance.
app	text	Client connected to the DN.
tid	bigint	Thread ID of the current thread.
lwtid	integer	Lightweight thread ID of the current thread.
query_id	bigint	Query ID of the current thread.
socket	integer	Socket FD of the current physical connection.
remote_ip	text	Peer IP address of the current connection.
remote_port	text	Peer port of the current connection.
local_ip	text	Local IP address of the current connection.
local_port	text	Local port of the current connection.

12.3.3.2 GS_GET_LISTEN_ADDRESS_EXT_INFO

GS_GET_LISTEN_ADDRESS_EXT_INFO queries the extended IP address configured for the current instance. Currently, this view cannot be queried.

Table 12-172 GS_GET_LISTEN_ADDRESS_EXT_INFO columns

Name	Type	Description
node_name	text	Name of the current instance.
host	text	Listening IP address of the current instance.
port	bigint	Listening port of the current instance.
ext_listen_ip	text	Extended IP address configured for the current instance.

12.3.3.3 PG_COMM_DELAY

PG_COMM_DELAY displays the communication library delay status for a single DN.

Table 12-173 PG_COMM_DELAY columns

Name	Type	Description
node_name	text	Node name
remote_name	text	Name of the peer node
remote_host	text	IP address of the peer node
stream_num	integer	Number of logical stream connections used by the current physical connection
min_delay	integer	Minimum delay of the current physical connection within 1 minute, in microsecond **NOTE** A negative result is invalid. Wait until the delay status is updated and query again.
average	integer	Average delay of the current physical connection within 1 minute, in microsecond
max_delay	integer	Maximum delay of the current physical connection within 1 minute, in microsecond

12.3.3.4 PG_COMM_RECV_STREAM

PG_COMM_RECV_STREAM displays the receiving stream status of all the communication libraries for a single DN.

Table 12-174 PG_COMM_RECV_STREAM columns

Name	Type	Description
node_name	text	Node name
local_tid	bigint	ID of the thread using this stream
remote_name	text	Name of the peer node
remote_tid	bigint	Peer thread ID
idx	integer	Peer DN ID in the local DN
sid	integer	Stream ID in the physical connection
tcp_sock	integer	TCP socket used in the stream
state	text	Status of the stream • **UNKNOWN**: The logical connection status is unknown. • **READY**: The logical connection is ready. • **RUN**: The logical connection sends packets normally. • **HOLD**: The logical connection is waiting to send packets. • **CLOSED**: The logical connection is closed. • **TO_CLOSED**: The logical connection will be closed.
query_id	bigint	**debug_query_id** corresponding to the stream
pn_id	integer	**plan_node_id** of the query executed by the stream
send_smp	integer	**smpid** of the sender of the query executed by the stream
recv_smp	integer	**smpid** of the receiver of the query executed by the stream
recv_bytes	bigint	Total data volume received from the stream, in byte
time	bigint	Current lifecycle service duration of the stream, in ms
speed	bigint	Average receiving rate of the stream, in byte/s
quota	bigint	Current communication quota value of the stream, in byte
buff_usize	bigint	Current size of the data cache of the stream, in byte

12.3.3.5 PG_COMM_SEND_STREAM

PG_COMM_SEND_STREAM displays the sending stream status of all the communication libraries for a single DN.

Table 12-175 PG_COMM_SEND_STREAM columns

Name	Type	Description
node_name	text	Node name
local_tid	bigint	ID of the thread using this stream
remote_name	text	Name of the peer node
remote_tid	bigint	Peer thread ID
idx	integer	Peer DN ID in the local DN
sid	integer	Stream ID in the physical connection
tcp_sock	integer	TCP socket used in the stream
state	text	Status of the stream • **UNKNOWN**: The logical connection status is unknown. • **READY**: The logical connection is ready. • **RUN**: The logical connection sends packets normally. • **HOLD**: The logical connection is waiting to send packets. • **CLOSED**: The logical connection is closed. • **TO_CLOSED**: The logical connection will be closed.
query_id	bigint	**debug_query_id** corresponding to the stream
pn_id	integer	**plan_node_id** of the query executed by the stream
send_smp	integer	**smpid** of the sender of the query executed by the stream
recv_smp	integer	**smpid** of the receiver of the query executed by the stream
send_bytes	bigint	Total data volume sent by the stream, in byte
time	bigint	Current lifecycle service duration of the stream, in ms

Name	Type	Description
speed	bigint	Average sending rate of the stream, in byte/s
quota	bigint	Current communication quota value of the stream, in byte
wait_quota	bigint	Extra time generated when the stream waits the quota value, in ms

12.3.3.6 PG_COMM_STATUS

PG_COMM_STATUS displays the communications library status for a single DN.

Table 12-176 PG_COMM_STATUS columns

Name	Type	Description
node_name	text	Node name.
rxpck_rate	integer	Receiving rate of the communications library on the node, in byte/s.
txpck_rate	integer	Sending rate of the communications library on the node, in byte/s.
rxkbyte_rate	bigint	Receiving rate of the communications library on the node, in kbyte/s.
txkbyte_rate	bigint	Sending rate of the communications library on the node, in kbyte/s.
buffer	bigint	Size of the buffer of the Cmailbox.
memkbyte_libcomm	bigint	Communication memory size of the **libcomm** thread, in bytes.
memkbyte_libpq	bigint	Communication memory size of the **libpq** thread, in bytes.
used_pm	integer	Real-time usage of the **postmaster** thread.
used_sflow	integer	Real-time usage of the **gs_sender_flow_controller** thread.
used_rflow	integer	Real-time usage of the **gs_receiver_flow_controller** thread.
used_rloop	integer	Highest real-time usage among multiple **gs_receivers_loop** threads.
stream	integer	Total number of used logical connections.

12.3.3.7 PG_GET_INVALID_BACKENDS

The view cannot be queried, and the error message "Un-support feature" is displayed.

12.3.4 Segment-Page Storage

12.3.4.1 GS_SEG_DATAFILES

GS_SEG_DATAFILES displays information about data files in all tablespaces. Only an administrator can query the information.

Table 12-177 GS_SEG_DATAFILES columns

Name	Type	Description
node_name	text	Node name.
file_name	text	Data file name, for example, **base/17467/2_fsm**.
file_id	integer	Data file ID.
bucketnode	integer	• **0** to **1023** indicate the bucket nodes of a hash bucket table. • **1024** indicates the bucket node of a segment-page ordinary table. • **1025** indicates the bucket node of a segment-page global temporary table. • **1026** indicates the bucket node of a segment-page unlogged table. • **1027** indicates the bucket node of a segment-page local temporary table.
forknum	integer	Segment fork type. Value range: • **0**: main fork. • **1**: fsm fork • **2**: vm fork
tablespace_name	name	Name of the tablespace to which a data file belongs.
contents	text	Storage content of a data file. Value range: • **permanent**: permanent • **unlogged**: no log • **temporary**: global temporary • **temporary2**: local temporary

Name	Type	Description
extent_size	integer	Extent size of a data file.
meta_blocks	bigint	Number of allocated metadata pages of a data file.
data_blocks	bigint	Number of allocated data pages of a data file.
total_blocks	bigint	Total number of physical pages in a data file.
high_water_mark	bigint	High watermark of the number of pages used by a data file.
utilization	real	Percentage of used blocks to the total number of blocks, that is, (**data_blocks** + **meta_blocks**)/ **total_blocks**.

12.3.4.2 GS_SEG_DATAFILE_LAYOUT

GS_SEG_DATAFILE_LAYOUT displays the static layout of data files 1 to 5 in segment-page mode. Only an administrator can query the information.

Table 12-178 GS_SEG_DATAFILE_LAYOUT columns

Name	Type	Description
version	text	Segment-page version. Default value: **1.0**.
seg_storage_type	text	Type of the data to be stored. Value range: • **segment** indicates common segment-page data. • **hashbucket** indicates hash bucket data.
file_id	integer	Data file ID.
section_id	integer	Data section ID of a data file.

Name	Type	Description
section_type	text	Type of the data file section. Value range: • **file_header** indicates the file header. • **spc_header** indicates the space header. • **map_header** indicates the mapping header. • **map_pages** indicates the mapping page. • **ip_pages(inverse pointer pages)** indicates the reverse pointer page. • **data_pages** indicates the data page.
page_start	bigint	Start page number of the data section.
page_end	bigint	End page number of the data section.
page_count	bigint	Number of pages in the data section.
total_size	bigint	Size of the data section. The unit is byte.

12.3.4.3 GS_SEG_EXTENTS

GS_SEG_EXTENTS displays extent information about all tablespaces. This view displays all extents of user segments, including segment heads, fork heads, and level-1 pages in file 1 and data extents in files 2 to 5. Only an administrator can query the information.

Table 12-179 GS_SEG_EXTENTS columns

Name	Type	Description
node_name	text	Node name.
tablespace_name	name	Tablespace to which a segment belongs.

Name	Type	Description
bucketnode	integer	• **0** to **1023** indicate the bucket nodes of a hash bucket table. • **1024** indicates the bucket node of a segment-page ordinary table. • **1025** indicates the bucket node of a segment-page global temporary table. • **1026** indicates the bucket node of a segment-page unlogged table. • **1027** indicates the bucket node of a segment-page local temporary table.
head_block_id	bigint	Page number of the segment header.
extent_id	integer	Logical extent number.
file_id	integer	ID of the data file where the extent is located.
forknum	integer	Fork type of a segment object. Value range: • **0**: main fork. • **1**: fsm fork. • **2**: vm fork.
block_id	bigint	Start page number in the data file where the extent is located.
blocks	integer	Extent size. The value can be **1**, **8**, **128**, **1024**, or **4096**.
usage_type	text	Usage type of the extension. Value range: • **segment head** indicates the segment header. • **fork head** indicates the fork header. • **level1 page** indicates the level-1 page. • **data extent** indicates data extents.

12.3.4.4 GS_SEG_SEGMENTS

GS_SEG_SEGMENTS displays information about segments in all tablespaces, including tables, indexes, TOAST, TOAST INDEX segments, and fsm fork and vm fork segments. Only an administrator can query the information.

Table 12-180 GS_SEG_SEGMENTS columns

Name	Type	Description
node_name	text	Node name.
schema_name	name	Namespace to which a segment belongs.
segment_name	name	Segment name. Source: pg_class and pg_partition.relname.
partition_name	name	Partition name of the segment object; **NULL** if the segment object is not partitioned. Source: pg_partition.relname.
forknum	integer	Fork type of a segment object. Value range: ● **0**: main fork. ● **1**: fsm fork. ● **2**: vm fork.
segment_type	text	Segment object type. Value range: ● **table**: segment-page ordinary table. ● **table partition**: segment-page partitioned table (main table and child table) and segment-page level-2 partitioned table (level-1 partitioned table). ● **table subpartition**: segment-page level-2 partitioned table (top node table and level-2 partitioned table). ● **index**: index of a segment-page ordinary table. ● **index partition**: index of a segment-page partitioned table or level-2 partitioned table. ● **global partition index**: global index for segment-page partitioned tables and level-2 partitioned tables. ● **toast**: segment-page TOAST table. ● **toast index**: index of a segment-page TOAST table.
tablespace_name	name	Tablespace to which a segment belongs.

Name	Type	Description
bucketnode	integer	• **0** to **1023** indicate the bucket nodes of a hash bucket table. • **1024** indicates the bucket node of a segment-page ordinary table. • **1025** indicates the bucket node of a segment-page global temporary table. • **1026** indicates the bucket node of a segment-page unlogged table. • **1027** indicates the bucket node of a segment-page local temporary table.
head_block_id	bigint	Page number of the segment header.
contents	text	Storage content of a data file. Value range: • **permanent**: permanent. • **unlogged**: no log. • **temporary**: global temporary. • **temporary2**: local temporary.
table_name	name	Name of the base table to which a segment belongs.
blocks	bigint	Number of logical pages of a segment.
total_blocks	bigint	Number of physical pages of a segment.
extents	integer	Number of logical extents of a segment.
total_extents	integer	Number of physical extents of a segment.
head_lsn	text	Segment header LSN.
level0_slots	bigint[]	Level-0 slot array of segment extent mapping.
level1_slots	bigint[]	Level-1 slot array of segment extent mapping.
fork_head	bigint[]	Fork head array of a segment.

12.3.4.5 GS_SEG_SEGMENT_LAYOUT

GS_SEG_SEGMENT_LAYOUT displays the static layout of segment-page files. Only the administrator can query the information.

Table 12-181 GS_SEG_SEGMENT_LAYOUT columns

Name	Type	Description
version	text	Segment-page version. Default value: **1.0**.
section_id	integer	ID of the data section divided by a segment.
section_type	text	Extent type of the segment data section. Value range: • **meta**: segment header. • **data**: data.
extent_size	integer	Extent size. The unit is byte.
extent_page_count	integer	Number of extent pages.
extent_count_start	bigint	Start extent number.
extent_count_end	bigint	End extent number.
total_size	bigint	Size of the segment data section. The unit is byte.

12.3.4.6 GS_SEG_SPC_EXTENTS

GS_SEG_SPC_EXTENTS displays information about the used extents of all tablespaces. The output contains segment heads, fork heads, level-1 pages, data extents. Only the administrator can query the information.

Table 12-182 GS_SEG_SPC_EXTENTS

Name	Type	Description
node_name	text	Node name.
tablespace_name	name	Tablespace name.
file_id	integer	Data file ID.
bucketnode	integer	• **0** to **1023** indicate the bucket nodes of a hash bucket table. • **1024** indicates the bucket node of a segment-page ordinary table. • **1025** indicates the bucket node of a segment-page global temporary table. • **1026** indicates the bucket node of a segment-page unlogged table. • **1027** indicates the bucket node of a segment-page local temporary table.

Name	Type	Description
forknum	integer	Fork type of a segment object. The options are as follows: • **0**: main fork • **1**: fsm fork • **2**: vm fork
block_id	bigint	Start page number of a data extent.
blocks	integer	Data extension size. The value can be **1**, **8**, **128**, **1024**, or **4096**.
contents	text	Storage content of a data file. The options are as follows: • **permanent**: permanent • **unlogged**: no log • **temporary**: global temporary • **temporary2**: local temporary
in_used	text	Specifies whether a resource has been allocated. The value can be **Y** or **N**.
mapblock_location	text	Position of the extension in the map block. The format is (*page_id*, *offset*).
head_file_id	integer	Segment header file ID.
head_block_id	bigint	Page number of the segment header.
usage_type	text	Extended usage type. The options are as follows: • **segment head** indicates the segment header. • **fork head** indicates the fork header. • **level1 page** indicates the level-1 page. • **data extent** indicates data extents.
remain_flag	text	Specifies whether it is a shrink residual extension. The value can be **Y** or **N**.
special_data	integer	Special data section of the reverse pointer corresponding to an extent.
ipblock_location	text	Position of the extended reverse pointer. The format is (*block_id*, *offset*).

12.3.4.7 GS_SEG_SPC_SEGMENTS

GS_SEG_SPC_SEGMENTS displays information about used segments in all tablespaces. Only the administrator can query the information.

Table 12-183 GS_SEG_SPC_SEGMENTS columns

Name	Type	Description
node_name	text	Node name.
tablespace_name	name	Tablespace name.
file_id	integer	Data file ID.
bucketnode	integer	• **0** to **1023** indicate the bucket nodes of a hash bucket table. • **1024** indicates the bucket node of a segment-page ordinary table. • **1025** indicates the bucket node of a segment-page global temporary table. • **1026** indicates the bucket node of a segment-page unlogged table. • **1027** indicates the bucket node of a segment-page local temporary table.
forknum	integer	Fork type of a segment object. The options are as follows: • **0**: main fork • **1**: fsm fork • **2**: vm fork
block_id	bigint	Start page number of a data extent.
blocks	integer	Data extension size. The value can be **1**, **8**, **128**, **1024**, or **4096**.
contents	text	Storage content of a data file. The options are as follows: • **permanent**: permanent • **unlogged**: no log • **temporary**: global temporary • **temporary2**: local temporary
in_used	text	Specifies whether a resource has been allocated. The value can be **Y** or **N**.
mapblock_location	text	Position of the extension in the map block. The format is (*page_id*, *offset*).
head_file_id	integer	Segment header file ID.
head_block_id	bigint	Page number of the segment header.

Name	Type	Description
usage_type	text	Extended usage type. The options are as follows: • **segment head** indicates the segment header. • **fork head** indicates the fork header. • **level1 page** indicates the level-1 page. • **data extent** indicates data extents.
remain_flag	text	Specifies whether it is a shrink residual extension. The value can be **Y** or **N**.
special_data	integer	Special data area of the reverse pointer corresponding to an extent.
ipblock_location	text	Position of the extended reverse pointer. The format is (*block_id*, *offset*).

12.3.4.8 GS_SEG_SPC_REMAIN_EXTENTS

GS_SEG_REMAIN_EXTENTS displays the residual isolated extent information of the index tablespace. The extent information is not included in the residual segment and can be used as an independent clearing unit. Only the administrator can query the information.

Table 12-184 GS_SEG_SPC_REMAIN_EXTENTS columns

Name	Type	Description
node_name	text	Node name.
tablespace_name	name	Tablespace name.
file_id	integer	Data file ID.
bucketnode	integer	• **0** to **1023** indicate the bucket nodes of a hash bucket table. • **1024** indicates the bucket node of a segment-page ordinary table. • **1025** indicates the bucket node of a segment-page global temporary table. • **1026** indicates the bucket node of a segment-page unlogged table. • **1027** indicates the bucket node of a segment-page local temporary table.

Name	Type	Description
forknum	integer	Fork type of a segment object. The options are as follows: • **0**: main fork • **1**: fsm fork • **2**: vm fork
block_id	bigint	Start page number of a data extent.
blocks	integer	Data extension size. The value can be **1**, **8**, **128**, **1024**, or **4096**.
contents	text	Storage content of a data file. The options are as follows: • **permanent**: permanent • **unlogged**: no log • **temporary**: global temporary • **temporary2**: local temporary
in_used	text	Specifies whether a resource has been allocated. The value can be **Y** or **N**.
mapblock_location	text	Position of the extension in the map block. The format is (*page_id*, *offset*).
head_file_id	integer	Segment header file ID.
head_block_id	bigint	Page number of the segment header.
usage_type	text	Extended usage type. The options are as follows: • **segment head** indicates the segment header. • **fork head** indicates the fork header. • **level1 page** indicates the level-1 page. • **data extent** indicates data extents.
remain_flag	text	Specifies whether it is a shrink residual extension. The value can be **Y** or **N**.
special_data	integer	Special data area of the reverse pointer corresponding to an extent.
ipblock_location	text	Position of the extended reverse pointer. The format is (*block_id*, *offset*).

12.3.4.9 GS_SEG_SPC_REMAIN_SEGMENTS

GS_SEG_SPC_REMAIN_SEGMENTS displays the residual segment information of the index tablespace, including the main fork, fsm fork, and vm fork segments. Only the administrator can query the information.

Table 12-185 GS_SEG_SPC_REMAIN_SEGMENTS columns

Name	Type	Description
node_name	text	Node name.
tablespace_name	name	Tablespace name.
file_id	integer	Data file ID.
bucketnode	integer	• **0** to **1023** indicate the bucket nodes of a hash bucket table. • **1024** indicates the bucket node of a segment-page ordinary table. • **1025** indicates the bucket node of a segment-page global temporary table. • **1026** indicates the bucket node of a segment-page unlogged table. • **1027** indicates the bucket node of a segment-page local temporary table.
forknum	integer	Fork type of a segment object. The options are as follows: • **0**: main fork • **1**: fsm fork • **2**: vm fork
block_id	bigint	Start page number of a data extent.
blocks	integer	Data extension size. The value can be **1**, **8**, **128**, **1024**, or **4096**.
contents	text	Storage content of a data file. The options are as follows: • **permanent**: permanent • **unlogged**: no log • **temporary**: global temporary • **temporary2**: local temporary
in_used	text	Specifies whether a resource has been allocated. The value can be **Y** or **N**.
mapblock_location	text	Position of the extension in the map block. The format is (*page_id*, *offset*).
head_file_id	integer	Segment header file ID.
head_block_id	bigint	Page number of the segment header.

Name	Type	Description
usage_type	text	Extended usage type. The options are as follows: • **segment head** indicates the segment header. • **fork head** indicates the fork header. • **level1 page** indicates the level-1 page. • **data extent** indicates data extents.
remain_flag	text	Specifies whether it is a shrink residual extension. The value can be **Y** or **N**.
special_data	integer	Special data area of the reverse pointer corresponding to an extent.
ipblock_location	text	Position of the extended reverse pointer. The format is (*block_id*, *offset*).

12.3.5 SPM

12.3.5.1 GS_SPM_SQL_BASELINE

GS_SPM_SQL_BASELINE is used to view the baseline information of the current user. Users with permissions higher than common users can access this view.

Table 12-186 GS_SPM_SQL_BASELINE columns

Name	Type	Description
sql_namespace	oid	OID of a schema.
sql_hash	bigint	Unique ID of an SQL statement in the current schema.
plan_hash	bigint	Unique ID of a plan in the current SQL statement.
outline	text	Outline text, which can be used to fix a group of hints of the current plan.
status	text	Status of a plan. Value range: • **UNACC**: The plan is not accepted. • **ACC**: The plan has been accepted. • **FIXED**: indicates a special ACC plan. The matching priority of this plan is higher than that of other ACC plans.
gplan	Boolean	Specifies whether the plan corresponding to the current outline is a gplan.

Name	Type	Description
cost	double precision	Total plan cost.
sql_text	text	SQL text string.
param_num	integer	Number of SQL parameters.
source	text	Baseline source.
creation_time	timestamp with time zone	Time when the baseline is created.
modification_time	timestamp with time zone	Time when the baseline is modified.
jump_intercept_cnt	bigint	Number of baseline interception plan jumps.
invalid	Boolean	Specifies whether the current baseline is invalid.

12.3.5.2 GS_SPM_SQL_PARAM

GS_SPM_SQL_PARAM is used to view the SQL parameter information of the current user. Users with permissions higher than common users can access this view.

Table 12-187 GS_SPM_SQL_PARAM columns

Name	Type	Description
sql_namespace	oid	OID of a schema.
sql_hash	bigint	Unique ID of an SQL statement in the current schema.
sql_text	text	SQL text string.
position	integer	Position index of a parameter in the SQL statement, starting from 0.
datatype	integer	OID of the parameter type.
datatype_string	text	Character string of a parameter type.
value_string	text	Character string of a parameter value.
is_null	Boolean	Specifies whether a parameter value is **NULL**.
hash_value	bigint	Hash value of a parameter value.

Name	Type	Description
creation_time	timestamp with time zone	Time when a record is created.

12.3.5.3 GS_SPM_SQL_EVOLUTION

GS_SPM_SQL_EVOLUTION is used to view the planned evolution result of the current user. Common users and users with higher permissions can access this view.

Table 12-188 GS_SPM_SQL_EVOLUTION columns

Name	Type	Description
sql_namespace	oid	OID of a schema.
sql_hash	bigint	Unique ID of an SQL statement in the current schema.
plan_hash	bigint	Unique ID of a plan in the current SQL statement.
better	Boolean	Determines whether it is a positive evolution.
status	text	Specifies whether exceptions occur during the evolution. Value range: • **SUCCESS**: The evolution is successful. • **FAILED**: The evolution fails.
refer_plan	bigint	Plan hash used as a reference for report generation.
sql_text	text	SQL text string.
outline	text	Hint string list of the current plan.
reason	text	Content of the evolution report.
gplan	Boolean	Determines whether the plan is a gplan.
creation_time	timestamp with time zone	Time when the evolution result is created.

12.3.5.4 GS_SPM_SYS_BASELINE

GS_SPM_SYS_BASELINE is used to view baseline information in the database. The user with the SYSADMIN permission and initial users can access this view.

Table 12-189 GS_SPM_SYS_BASELINE columns

Name	Type	Description
sql_hash	bigint	Unique ID of the SQL statement in the SPM.
plan_hash	bigint	Unique ID of a plan in the current SQL statement.
unique_sql_id	bigint	Unique ID of an SQL statement in the database.
outline	text	Outline text, which can be used to fix a group of hints of the current plan.
status	integer	Plan status. The options are as follows: • **0** (UNACC): The plan is not accepted. • **1** (ACC): The plan is accepted. • **2** (FIXED): a special ACC plan. The matching priority of this plan is higher than that of other ACC plan.
gplan	Boolean	Specifies whether the plan corresponding to the current outline is a gplan.
cost	double precision	Total plan cost.
sql_text	text	SQL text string.
param_num	integer	Number of SQL parameters.
source	text	Baseline source.
creation_time	timestamp with time zone	Time when the baseline is created.
modification_time	timestamp with time zone	Time when the baseline is modified.
jump_intercept_cnt	bigint	Number of baseline interception plan jumps.
invalid	Boolean	Specifies whether the current baseline is invalid.

12.3.6 Auditing

12.3.6.1 ADM_AUDIT_OBJECT

ADM_AUDIT_OBJECT displays the audit trail records of all objects in the database. This view exists in both PG_CATALOG and SYS schema. By default, only the system

administrator can access this view. Common users can access the view only after being authorized.

Table 12-190 ADM_AUDIT_OBJECT columns

Name	Type	Description
os_username	character varying(255)	Not supported. Its value is **NULL**.
username	character varying(128)	Name of the user whose operation is audited, not the user ID.
userhost	character varying(128)	Not supported. Its value is **NULL**.
terminal	character varying(255)	Not supported. Its value is **NULL**.
timestamp	timestamp(0) without time zone	Date and time when an audit trail entry is created in the local database session time zone (user login date and time of the entry created by the audit session)
owner	character varying(128)	Creator of the object affected by the operation.
obj_name	character varying(128)	Name of the object affected by the operation.
action_name	character varying(28)	Action name corresponding to the numeric code in the **ACTION** column in DBA_AUDIT_TRAIL **NOTE** The **action_name** column in GaussDB is inconsistent with the audit action of database A.
new_owner	character varying(128)	Not supported. Its value is **NULL**.
new_name	character varying(128)	Not supported. Its value is **NULL**.
ses_actions	character varying(19)	Not supported. Its value is **NULL**.

Name	Type	Description
comment_text	character varying(4000)	Not supported. Its value is **NULL**.
sessionid	numeric	Not supported. Its value is **NULL**.
entryid	numeric	Not supported. Its value is **NULL**.
statementid	numeric	Not supported. Its value is **NULL**.
returncode	numeric	Not supported. Its value is **NULL**.
priv_used	character varying(40)	Not supported. Its value is **NULL**.
client_id	character varying(128)	Not supported. Its value is **NULL**.
econtext_id	character varying(64)	Not supported. Its value is **NULL**.
session_cpu	numeric	Not supported. Its value is **NULL**.
extended_timestamp	timestamp(6) with time zone	Timestamp when an audit trail entry is created (user login timestamp of the entry created by the audit session in UTC).
proxy_sessionid	numeric	Not supported. Its value is **NULL**.
global_uid	character varying(32)	Not supported. Its value is **NULL**.
instance_number	numeric	Not supported. Its value is **NULL**.
os_process	character varying(16)	Not supported. Its value is **NULL**.
transactionid	text	Identifier of the transaction that accesses or modifies an object. **NOTE** The type of the **transactionid** column in GaussDB must be the same as that in database A.
scn	numeric	Not supported. Its value is **NULL**.
sql_bind	nvarchar2(2000)	Not supported. Its value is **NULL**.

Name	Type	Description
sql_text	character varying	SQL text of the query. **NOTE** The **sql_text** column in GaussDB is the parsed SQL description statement, which is not completely the same as the executed SQL statement.
obj_edition_name	character varying(128)	Not supported. Its value is **NULL**.

12.3.6.2 ADM_AUDIT_SESSION

ADM_AUDIT_SESSION displays all audit trail records concerning CONNECT and DISCONNECT. The audit information of GaussDB is mainly queried through the pg_query_audit function. This view exists in both PG_CATALOG and SYS schema. Only users with the AUDITADMIN attribute can view audit information.

Table 12-191 ADM_AUDIT_SESSION columns

Name	Type	Description
os_username	character varying(255)	Not supported. Its value is **NULL**.
username	character varying(128)	Name of the user whose operation is audited, not the user ID.
userhost	character varying(128)	Not supported. Its value is **NULL**.
terminal	character varying(255)	Not supported. Its value is **NULL**.
timestamp	timestamp(0) without time zone	Date and time when an audit trail entry is created (user login date and time of the entry created by the audit session)
action_name	character varying(28)	Action name corresponding to the numeric code in the **ACTION** column in DBA_AUDIT_TRAIL. **NOTE** The **action_name** column in GaussDB is inconsistent with the audit action of database A.

Name	Type	Description
logoff_time	timestamp(0) without time zone	Not supported. Its value is **NULL**.
logoff_lread	numeric	Not supported. Its value is **NULL**.
logoff_pread	numeric	Not supported. Its value is **NULL**.
logoff_lwrite	numeric	Not supported. Its value is **NULL**.
logoff_dlock	character varying(40)	Not supported. Its value is **NULL**.
sessionid	numeric	Not supported. Its value is **NULL**.
returncode	numeric	Not supported. Its value is **NULL**.
client_id	character varying(128)	Not supported. Its value is **NULL**.
session_cpu	numeric	Not supported. Its value is **NULL**.
extended_timestamp	timestamp(6) with time zone	Time zone of the timestamp when an audit trail entry is created (user login date and time of the entry created by the audit session in UTC)
proxy_sessionid	numeric	Not supported. Its value is **NULL**.
global_uid	character varying(32)	Not supported. Its value is **NULL**.
instance_number	numeric	Not supported. Its value is **NULL**.
os_process	character varying(16)	Not supported. Its value is **NULL**.

12.3.6.3 ADM_AUDIT_STATEMENT

ADM_AUDIT_STATEMENT displays all grant and revoke audit trail entries. The audit information of GaussDB is mainly queried through the pg_query_audit function. This view exists in both PG_CATALOG and SYS schema. Only users with the AUDITADMIN attribute can view audit information.

Table 12-192 ADM_AUDIT_STATEMENT columns

Name	Type	Description
os_username	character varying(255)	Not supported. Its value is **NULL**.
username	character varying(128)	Name of the user whose operation is audited, not the user ID.
userhost	character varying(128)	Not supported. Its value is **NULL**.
terminal	character varying(255)	Not supported. Its value is **NULL**.
timestamp	timestamp(0) without time zone	Date and time when an audit trail entry is created (user login date and time of the entry created by AUDIT SESSION).
owner	character varying(128)	Not supported. Its value is **NULL**.
obj_name	character varying(128)	Name of the object affected by the operation.
action_name	character varying(28)	Action name corresponding to the numeric code in the **ACTION** column in **DBA_AUDIT_TRAIL**. **NOTE** The **action_name** column of GaussDB is inconsistent with the audit action of database A, and the **transactionid** column is consistent with the type of **transactionid** data in database A.
new_name	character varying(128)	Not supported. Its value is **NULL**.
obj_privilege	character varying(32)	Not supported. Its value is **NULL**.
sys_privilege	character varying(40)	Not supported. Its value is **NULL**.
admin_option	character varying(1)	Not supported. Its value is **NULL**.

Name	Type	Description
grantee	character varying(128)	Not supported. Its value is **NULL**.
audit_option	character varying(40)	Not supported. Its value is **NULL**.
ses_actions	character varying(19)	Not supported. Its value is **NULL**.
comment_text	character varying(4000)	Not supported. Its value is **NULL**.
sessionid	numeric	Not supported. Its value is **NULL**.
entryid	numeric	Not supported. Its value is **NULL**.
statementid	numeric	Not supported. Its value is **NULL**.
returncode	numeric	Not supported. Its value is **NULL**.
priv_used	character varying(40)	Not supported. Its value is **NULL**.
client_id	character varying(128)	Not supported. Its value is **NULL**.
econtext_id	character varying(64)	Not supported. Its value is **NULL**.
session_cpu	numeric	Not supported. Its value is **NULL**.
extended_timestamp	timestamp(6) with time zone	Timestamp when an audit trail entry is created (user login timestamp of the entry created by AUDIT SESSION in UTC).
proxy_sessionid	numeric	Not supported. Its value is **NULL**.
global_uid	character varying(32)	Not supported. Its value is **NULL**.
instance_number	numeric	Not supported. Its value is **NULL**.
os_process	character varying(16)	Not supported. Its value is **NULL**.

Name	Type	Description
transactionid	text	Identifier of the transaction that accesses or modifies an object. **NOTE** The type of the **transactionid** column in GaussDB must be the same as that in database A.
scn	numeric	Not supported. Its value is **NULL**.
sql_bind	nvarchar2(2000)	Not supported. Its value is **NULL**.
sql_text	character varying(2000)	SQL text of the query. **NOTE** The **sql_text** column in GaussDB is the parsed SQL description statement, which is not completely the same as the executed SQL statement.
obj_edition_name	character varying(128)	Not supported. Its value is **NULL**.

12.3.6.4 ADM_AUDIT_TRAIL

ADM_AUDIT_TRAIL displays all standard audit trail entries. The audit information of GaussDB is mainly queried through the pg_query_audit function. This view exists in both PG_CATALOG and SYS schema. Only users with the AUDITADMIN attribute and the SELECT permission on ADM_AUDIT_TRAIL can view audit information. If separation of duties is disabled, users with the SYSADMIN attribute can also view audit information.

Table 12-193 ADM_AUDIT_TRAIL columns

Name	Type	Description
os_username	character varying(255)	Not supported. Its value is **NULL**.
username	character varying(128)	Name of the user whose operation is audited, not the user ID.
userhost	character varying(128)	Not supported. Its value is **NULL**.
terminal	character varying(255)	Not supported. Its value is **NULL**.
timestamp	timestamp(0) without time zone	Date and time when an audit trail entry is created in the local database session time zone (user login date and time of the entry created by the audit session)

Name	Type	Description
owner	character varying(128)	Creator of the object affected by the operation.
obj_name	character varying(128)	Name of the object affected by the operation.
action	numeric	Not supported. Its value is **NULL**.
action_name	character varying(28)	Action type corresponding to the code in the action column. **NOTE** The **action_name** column in GaussDB is inconsistent with the audit action of database A.
new_owner	character varying(128)	Not supported. Its value is **NULL**.
new_name	character varying(128)	Not supported. Its value is **NULL**.
obj_privilege	character varying(32)	Not supported. Its value is **NULL**.
sys_privilege	character varying(40)	Not supported. Its value is **NULL**.
admin_option	character varying(1)	Not supported. Its value is **NULL**.
grantee	character varying(128)	Not supported. Its value is **NULL**.
audit_option	character varying(40)	Not supported. Its value is **NULL**.
ses_actions	character varying(19)	Not supported. Its value is **NULL**.
logoff_time	timestamp(0) without time zone	Not supported. Its value is **NULL**.
logoff_lread	numeric	Not supported. Its value is **NULL**.
logoff_pread	numeric	Not supported. Its value is **NULL**.
logoff_lwrite	numeric	Not supported. Its value is **NULL**.
logoff_dlock	character varying(40)	Not supported. Its value is **NULL**.
comment_text	character varying(4000)	Not supported. Its value is **NULL**.
sessionid	numeric	Not supported. Its value is **NULL**.

Name	Type	Description
entryid	numeric	Not supported. Its value is **NULL**.
statementid	numeric	Not supported. Its value is **NULL**.
returncode	numeric	Not supported. Its value is **NULL**.
priv_used	character varying(40)	Not supported. Its value is **NULL**.
client_id	character varying(128)	Not supported. Its value is **NULL**.
econtext_id	character varying(64)	Not supported. Its value is **NULL**.
session_cpu	numeric	Not supported. Its value is **NULL**.
extended_timestamp	timestamp(6) with time zone	Time zone of the timestamp when an audit trail entry is created (user login date and time of the entry created by the audit session in UTC).
proxy_sessionid	numeric	Not supported. Its value is **NULL**.
global_uid	character varying(32)	Not supported. Its value is **NULL**.
instance_number	numeric	Not supported. Its value is **NULL**.
os_process	character varying(16)	Not supported. Its value is **NULL**.
transactionid	text	Identifier of the transaction that accesses or modifies an object. **NOTE** The type of the **transactionid** column in GaussDB must be the same as that in database A.
scn	numeric	Not supported. Its value is **NULL**.
sql_bind	nvarchar2(2000)	Not supported. Its value is **NULL**.
sql_text	nvarchar2	SQL text of the query. **NOTE** The **sql_text** column in GaussDB is the parsed SQL description statement, which is not completely the same as the executed SQL statement.
obj_edition_name	character varying(128)	Not supported. Its value is **NULL**.
dbid	numeric	Not supported. Its value is **NULL**.
rls_info	clob	Not supported. Its value is **NULL**.

Name	Type	Description
current_user	character varying(128)	Not supported. Its value is **NULL**.

12.3.6.5 GS_AUDITING

GS_AUDITING displays all audit information about database-related operations. Only the users with system administrator or security policy administrator permission can access this view.

Table 12-194 GS_AUDITING columns

Name	Type	Description
polname	name	Policy name, which must be unique.
pol_type	text	Audit policy type. Value range: • **access**: DML operations are audited. • **privilege**: DDL operations are audited.
polenabled	Boolean	Specifies whether to enable a policy. • **t** (true): enabled. • **f** (false): disabled.
access_type	name	DML database operation types, such as SELECT, INSERT, and DELETE, or DDL database operation types, such as CREATE, ALTER, and DROP.
label_name	name	Resource label name. This column corresponds to the **polname** column in the GS_AUDITING_POLICY system catalog.
priv_object	text	Path of the database asset.
filter_name	text	Logical character string of a filter criterion

12.3.6.6 GS_AUDITING_ACCESS

GS_AUDITING_ACCESS displays all audit information about database DML-related operations. Only the users with system administrator or security policy administrator permission can access this view.

Table 12-195 GS_AUDITING_ACCESS columns

Name	Type	Description
polname	name	Policy name, which must be unique.

Name	Type	Description
pol_type	text	Audit policy type. The value **access** indicates that DML operations are audited.
polenabled	Boolean	Specifies whether the policy is enabled. • **t** (true): enabled. • **f** (false): disabled.
access_type	name	DML database operation type. For example, SELECT, INSERT, and DELETE.
label_name	name	Resource label name. This column corresponds to the **polname** column in the **GS_AUDITING_POLICY** system catalog.
access_object	text	Path of the database asset.
filter_name	text	Logical character string of a filter criterion

12.3.6.7 GS_AUDITING_PRIVILEGE

GS_AUDITING_PRIVILEGE displays all audit information about database DDL-related operations. Only the users with system administrator or security policy administrator permission can access this view.

Table 12-196 GS_AUDITING_PRIVILEGE columns

Name	Type	Description
polname	name	Policy name, which must be unique.
pol_type	text	Audit policy type. The value **privilege** indicates that DDL operations are audited.
polenabled	Boolean	Specifies whether the policy is enabled. • **t** (true): enabled. • **f** (false): disabled.
access_type	name	DDL database operation type. For example, CREATE, ALTER, and DROP.
label_name	name	Resource label name. This column corresponds to the **polname** column in the **GS_AUDITING_POLICY** system catalog.
priv_object	text	Full domain name of a database object.
filter_name	text	Logical character string of a filter criterion

12.3.7 User and Permission Management

12.3.7.1 ADM_COL_PRIVS

ADM_COL_PRIVS displays permission granting information about all columns. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG_CATALOG and SYS schema.

Table 12-197 ADM_COL_PRIVS columns

Name	Type	Description
grantor	character varying(128)	Name of the user who grants the permission.
owner	character varying(128)	Object owner.
grantee	character varying(128)	Name of the user or role to which the permission is granted.
table_schema	character varying(128)	Schema of an object
table_name	character varying(128)	Object name.
column_name	character varying(128)	Column name.
privilege	character varying(40)	Permission on a column.
grantable	character varying(3)	Specifies whether to grant privileges. • **YES**: The privileges are granted. • **NO**: The privileges are not granted.
common	character varying(3)	Not supported. Its value is **NULL**.
inherited	character varying(3)	Not supported. Its value is **NULL**.

12.3.7.2 ADM_ROLE_PRIVS

ADM_ROLE_PRIVS displays information about roles granted to all users and roles. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG_CATALOG and SYS schema.

Table 12-198 ADM_ROLE_PRIVS columns

Name	Type	Description
grantee	character varying(128)	Name of the user or role to which the permission is granted.
granted_role	character varying(128)	Name of the role to be granted.
admin_option	character varying(3)	Specifies whether the grant contains the **ADMIN** option. • **YES** • **NO**
delegate_option	character varying(3)	Not supported. Its value is **NULL**.
default_role	character varying(3)	Not supported. Its value is **NULL**.
os_granted	character varying(3)	Not supported. Its value is **NULL**.
common	character varying(3)	Not supported. Its value is **NULL**.
inherited	character varying(3)	Not supported. Its value is **NULL**.

12.3.7.3 ADM_ROLES

ADM_ROLES displays information about database roles. This view exists in both the PG_CATALOG and SYS schema. By default, only the system administrator can access this view.

Table 12-199 ADM_ROLES columns

Name	Type	Description
role	character varying(128)	Role name.
role_id	oid	Role ID.
authentication_type	text	Role authentication mechanism. • **password**: Password authentication is required. • **NULL**: Authentication is not required.

Name	Type	Description
common	character varying(3)	Not supported. Its value is **NULL**.
oracle_maintained	character varying(1)	Not supported. Its value is **NULL**.
inherited	character varying(3)	Not supported. Its value is **NULL**.
implicit	character varying(3)	Not supported. Its value is **NULL**.
external_name	character varying(4000)	Not supported. Its value is **NULL**.

12.3.7.4 ADM_SYS_PRIVS

ADM_SYS_PRIVS displays information about system permissions granted to users and roles. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG_CATALOG and SYS schema.

Table 12-200 ADM_SYS_PRIVS columns

Name	Type	Description
grantee	character varying(128)	Name of the user or role to which the permission is granted.
privilege	character varying(40)	System permissions or ANY permissions of the user. • System permissions include rolsuper, rolinherit, rolcreaterole, rolcreatedb, rolcatupdate, rolcanlogin, rolreplication, rolauditadmin, rolsystemadmin, roluseft, rolmonitoradmin, roloperatoradmin, and rolpolicyadmin. • For details about the value of the ANY permissions, see **Table 7-223**.
admin_option	character varying(3)	Specifies whether the grant contains the **ADMIN** option. • **YES** • **NO**
common	character varying(3)	Not supported. Its value is **NULL**.

Name	Type	Description
inherited	character varying(3)	Not supported. Its value is **NULL**.

12.3.7.5 ADM_TAB_PRIVS

ADM_TAB_PRIVS displays authorization information about all objects in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG_CATALOG and SYS schema.

Table 12-201 ADM_TAB_PRIVS columns

Name	Type	Description
grantee	character varying(128)	Name of the user or role to which the permission is granted.
owner	character varying(128)	Object owner.
table_name	character varying(128)	Object name.
grantor	character varying(128)	Name of the user who grants the permission.
privilege	character varying(40)	Permissions on an object, including USAGE, UPDATE, DELETE, INSERT, CONNECT, SELECT, and EXECUTE.
grantable	character varying(3)	Specifies whether the grant contains the **GRANT** option. • **YES** • **NO**

Name	Type	Description
type	character varying(24)	Object types, including NODE GROUP, COLUMN_ENCRYPTION_KEY, PACKAGE, COLUMN, TABLE, VIEW, SEQUENCE, TYPE, INDEX, DATABASE, DIRECTORY, FOREIGN DATA WRAPPER, FOREIGN SERVER, LANGUAGE, LARGE OBJECT, SCHEMA, TEMPLATE, FUNCTION, PROCEDURE, and TABLESPACE.
hierarchy	character varying(3)	Not supported. Its value is **NULL**.
common	character varying(3)	Not supported. Its value is **NULL**.
inherited	character varying(3)	Not supported. Its value is **NULL**.

12.3.7.6 ADM_USERS

ADM_USERS displays information about all database users. This view is accessible only to system administrators. This view exists in both PG_CATALOG and SYS schema.

Table 12-202 ADM_USERS columns

Name	Type	Description
username	character varying(128)	Username.
user_id	oid	User ID.

Name	Type	Description
account_status	character varying(32)	Account status. • **NULL**: The account is the initial system administrator with the highest permission. • **0**: normal. • **1**: The account is locked for a specific period of time because the number of failed login attempts exceeds the threshold. • **2**: The account is locked by the administrator.
lock_date	timestamp with time zone	By default, the creation date of the account is displayed. If the account is locked by the administrator or the account is locked because the number of login failures exceeds the threshold, the date when the account is locked is displayed. The value is **NULL** for the initial system administrator.
expiry_date	timestamp with time zone	Account expiration date.
default_tablespace	character varying(4000)	Default tablespace for storing data.
temporary_tablespace	character varying(4000)	Name of the default tablespace or tablespace group of a temporary table.
local_temp_tablespace	character varying(30)	Not supported. The default value is **NULL**.
created	timestamp with time zone	Date when a user is created.
profile	character varying(128)	Not supported. The default value is **NULL**.
initial_rsrc_consumer_group	character varying(128)	Not supported. The default value is **NULL**.
external_name	character varying(4000)	Not supported. The default value is **NULL**.
password_versions	character varying(12)	Encryption type of account passwords. Value range: **MD5**, **SHA256**, or **SM3**.
editions_enabled	character varying(1)	Not supported. The default value is **NULL**.

Name	Type	Description
authentication_type	text	Authentication mechanism of the user.
proxy_only_connect	character varying(1)	Not supported. The default value is **NULL**.
common	character varying(3)	Not supported. The default value is **NULL**.
last_login	timestamp with time zone	Last login.
oracle_maintained	character varying(1)	Not supported. The default value is **NULL**.
inherited	character varying(3)	Not supported. The default value is **NULL**.
default_collation	character varying(100)	Default collation of the user schema.
implicit	character varying(3)	Not supported. The default value is **NULL**.
all_shard	character varying(3)	Not supported. The default value is **NULL**.
password_change_date	timestamp with time zone	Date when the user password was set last time.

12.3.7.7 DB_COL_PRIVS

DB_COL_PRIVS displays the following granting information:

- Column permission granting information when the current user is the object owner, grantor, or grantee.
- Column permission granting information when the enabled role or PUBLIC role is the grantee.

By default, it is accessible to all users. This view exists in both PG_CATALOG and SYS schema.

Table 12-203 DB_COL_PRIVS columns

Name	Type	Description
grantor	character varying(128)	Name of the user who grants the permission.
owner	character varying(128)	Object owner.

Name	Type	Description
grantee	character varying(128)	Name of the user or role to which the permission is granted.
table_schema	character varying(128)	Schema of an object.
table_name	character varying(128)	Object name.
column_name	character varying(128)	Column name.
privilege	character varying(40)	Permission on a column.
grantable	character varying(3)	Specifies whether to grant privileges. • **YES** • **NO**
common	character varying(3)	Not supported. Its value is **NULL**.
inherited	character varying(3)	Not supported. Its value is **NULL**.

12.3.7.8 DB_DIRECTORIES

DB_DIRECTORIES is used to view all directories on which the current user has the operation permission. Administrators can query all directories, and common users can query only directories on which the current user has the operation permission. This view exists in both PG_CATALOG and SYS schema.

Table 12-204 DB_DIRECTORIES columns

Name	Type	Description
owner	oid	OID of the owner of a directory object.
directory_name	name	Name of a directory object.
directory_path	text	Directory path represented by a directory object.
origin_con_id	character varying(256)	ID of the container where a directory object is created. Not supported. Its value is **NULL**.

12.3.7.9 DB_TAB_PRIVS

DB_TAB_PRIVS displays authorization information about all objects accessible to the current user. All users can access this view. This view exists in both PG_CATALOG and SYS schema.

Table 12-205 DB_TAB_PRIVS columns

Name	Type	Description
grantor	character varying(128)	Name of the user who grants the permission.
grantee	character varying(128)	Name of the user or role to which the permission is granted.
table_schema	character varying(128)	Schema to which an object belongs.
table_name	character varying(128)	Object name.
privilege	character varying(40)	Permissions on an object, including USAGE, UPDATE, DELETE, INSERT, CONNECT, SELECT, and EXECUTE.
grantable	character varying(3)	Specifies whether the grant contains the **GRANT** option. • **YES** • **NO**
type	character varying(24)	Object types, including NODE GROUP, COLUMN_ENCRYPTION_KEY, PACKAGE, COLUMN, TABLE, VIEW, SEQUENCE, TYPE, INDEX, DATABASE, DIRECTORY, FOREIGN DATA WRAPPER, FOREIGN SERVER, LANGUAGE, LARGE OBJECT, SCHEMA, TEMPLATE, FUNCTION, PROCEDURE, and TABLESPACE.
hierarchy	character varying(3)	Not supported. Its value is **NULL**.
common	character varying(3)	Not supported. Its value is **NULL**.
inherited	character varying(3)	Not supported. Its value is **NULL**.

12.3.7.10 DB_USERS

DB_USERS displays all users of the database visible to the current user. However, it does not describe the users. By default, only the system administrator can access this view. This view exists in the PG_CATALOG and SYS schemas.

Table 12-206 DB_USERS columns

Name	Type	Description
user_id	oid	OID of a user
username	name	Username

12.3.7.11 GS_DB_PRIVILEGES

GS_DB_PRIVILEGES displays the granting of ANY permissions. Each record corresponds to a piece of authorization information.

Table 12-207 GS_DB_PRIVILEGES columns

Name	Type	Description
rolename	name	Username.
privilege_type	text	ANY permission of a user. For details about the value, see **Table 7-223**.
admin_option	text	Specifies whether the ANY permission recorded in the **privilege_type** column can be re-granted. • **yes** • **no**

12.3.7.12 GS_LABELS

GS_LABELS displays all configured resource labels. Only the users with system administrator or security policy administrator permission can access this view.

Table 12-208 GS_LABELS columns

Name	Type	Description
labelname	name	Resource label name
labeltype	name	Resource label type This parameter corresponds to the **labeltype** column in the **GS_POLICY_LABEL** system catalog.
fqdntype	name	Database resource type Such as table, schema, and index.
schemaname	name	Name of the schema to which the database resource belongs
fqdnname	name	Database resource name

Name	Type	Description
columnname	name	Name of the database resource column. If the marked database resource is not a column, this parameter is left blank.

12.3.7.13 MY_COL_PRIVS

MY_COL_PRIVS displays the column permission granting information of the current user as the object owner, grantor, or grantee. By default, it is accessible to all users. This view exists in both PG_CATALOG and SYS schema.

Table 12-209 MY_COL_PRIVS columns

Name	Type	Description
grantor	character varying(128)	Name of the user who grants the permission.
owner	character varying(128)	Object owner.
grantee	character varying(128)	Name of the user or role to which the permission is granted.
table_schema	character varying(128)	Schema of an object.
table_name	character varying(128)	Object name.
column_name	character varying(128)	Column name.
privilege	character varying(40)	Permission on a column.
grantable	character varying(3)	Specifies whether to grant privileges. • **YES** • **NO**
common	character varying(3)	Not supported. Its value is **NULL**.
inherited	character varying(3)	Not supported. Its value is **NULL**.

12.3.7.14 MY_ROLE_PRIVS

MY_ROLE_PRIVS displays permission information about roles (including the public role) granted to the current user. By default, it is accessible to all users. This view exists in both PG_CATALOG and SYS schema.

Table 12-210 MY_ROLE_PRIVS columns

Name	Type	Description
grantee	character varying(128)	Name of the user or role to which the permission is granted.
granted_role	character varying(128)	Role to be granted.
admin_option	character varying(3)	Specifies whether the grant contains the **ADMIN** option. • **YES** • **NO**
delegate_option	character varying(3)	Not supported. Its value is **NULL**.
default_role	character varying(3)	Not supported. Its value is **NULL**.
os_granted	character varying(3)	Not supported. Its value is **NULL**.
common	character varying(3)	Not supported. Its value is **NULL**.
inherited	character varying(3)	Not supported. Its value is **NULL**.

12.3.7.15 MY_SYS_PRIVS

MY_SYS_PRIVS displays information about system permissions granted to the current user. By default, it is accessible to all users. This view exists in both PG_CATALOG and SYS schema.

Table 12-211 MY_SYS_PRIVS columns

Name	Type	Description
grantee	character varying(128)	Name of the user or role to which the permission is granted.

Name	Type	Description
privilege	character varying(40)	System permissions or ANY permissions of the user. • System permissions include rolsuper, rolinherit, rolcreaterole, rolcreatedb, rolcatupdate, rolcanlogin, rolreplication, rolauditadmin, rolsystemadmin, roluseft, rolmonitoradmin, roloperatoradmin, and rolpolicyadmin. • For details about the value of the ANY permissions, see Table 7-223.
admin_option	character varying(3)	Specifies whether the grant contains the **ADMIN** option. • **YES** • **NO**
common	character varying(3)	Not supported. Its value is **NULL**.
inherited	character varying(3)	Not supported. Its value is **NULL**.

12.3.7.16 PG_RLSPOLICIES

PG_RLSPOLICIES displays information about row-level security policies. The initial user and users with the sysadmin attribute can view all policy information. Other users can view only the policy information in their own tables.

Table 12-212 PG_RLSPOLICIES columns

Name	Type	Description
schemaname	name	Name of the schema of the table object to which a row-level security policy is applied.
tablename	name	Name of the table object to which a row-level security policy is applied.

Name	Type	Description
policyname	name	Name of a row-level security policy.
policypermissive	text	Expression combination mode of a row-level security policy. The options are as follows: • **PERMISSIVE**: permissive policy, which is concatenated using the OR expression. • **RESTRICTIVE**: restrictive policy, which is concatenated using an AND expression.
policyroles	name[]	List of users affected by the row-level security policy. If this parameter is not specified, all users are affected.
policycmd	text	SQL operations affected by a row-level security policy.
policyqual	text	Expression of a row-level security policy.

12.3.7.17 PG_ROLES

PG_ROLES displays information about database roles. Initial users and users with the **sysadmin** or **createrole** attribute can view information about all roles. Other users can view only their own information.

Table 12-213 PG_ROLES columns

Name	Type	Reference	Description
rolname	name	N/A	Role name.
rolsuper	Boolean	N/A	Specifies whether a role is the initial system administrator with the highest permissions. • **t** (true): yes. • **f** (false): no.
rolinherit	Boolean	N/A	Specifies whether the role inherits the permissions for this type of roles. • **t** (true): yes. • **f** (false): no.
rolcreaterole	Boolean	N/A	Specifies whether the role can create other roles. • **t** (true): yes. • **f** (false): no.

Name	Type	Reference	Description
rolcreatedb	Boolean	N/A	Specifies whether the role can create databases. ● **t** (true): yes. ● **f** (false): no.
rolcatupdate	Boolean	N/A	Specifies whether the role can update system catalogs directly. Only the initial system administrator whose **usesysid** is **10** has this permission. This permission is unavailable for other users. ● **t** (true): yes. ● **f** (false): no.
rolcanlogin	Boolean	N/A	Specifies whether the role can log in to the database. ● **t** (true): yes. ● **f** (false): no.
rolreplication	Boolean	N/A	Specifies whether the role can be replicated. ● **t** (true): yes. ● **f** (false): no.
rolauditadmin	Boolean	N/A	Specifies whether the role is an audit administrator. ● **t** (true): yes. ● **f** (false): no.
rolsystemadmin	Boolean	N/A	Specifies whether the role is a system administrator. ● **t** (true): yes. ● **f** (false): no.
rolconnlimit	integer	N/A	Maximum number of concurrent connections that this role can initiate if this role can log in. The value **–1** indicates no limit.
rolpassword	text	N/A	Encrypted user password. The value is displayed as ********.
rolvalidbegin	timestamp with time zone	N/A	Start time for account validity (**NULL** if the start time is not specified).
rolvaliduntil	timestamp with time zone	N/A	End time for account validity (**NULL** if the end time is not specified).

Name	Type	Reference	Description
rolrespool	name	N/A	Resource pool that can be used by a user.
rolparentid	oid	**rolparentid** in **PG_AUTHID**	OID of a group user to which the user belongs
roltabspace	text	N/A	Storage space of the user permanent table, in KB.
rolconfig	text[]	**setconfig** in **PG_DB_ROLE_SETTING**	Default value of runtime configuration items.
oid	oid	**oid** in **PG_AUTHID**	Role ID
roluseft	Boolean	**roluseft** in **PG_AUTHID**	Specifies whether the role can perform operations on foreign tables • **t** (true): yes. • **f** (false): no.
rolkind	"char"	N/A	Role type • **n**: common user, that is, non-permanent user. • **p**: permanent user.
nodegroup	name	N/A	Unsupported currently.
roltempspace	text	N/A	Storage space of the user temporary table, in KB.
rolspillspace	text	N/A	Operator disk spill space of the user, in KB.
rolmonitoradmin	Boolean	N/A	Specifies whether the role is a monitor administrator. • **t** (true): yes. • **f** (false): no.
roloperatoradmin	Boolean	N/A	Specifies whether the role is an O&M administrator. • **t** (true): yes. • **f** (false): no.

Name	Type	Reference	Description
rolpolicyadmin	Boolean	N/A	Specifies whether the role is a security policy administrator. • **t** (true): yes. • **f** (false): no.

12.3.7.18 PG_SECLABELS

PG_SECLABELS provides information about storage security labels.

Table 12-214 PG_SECLABELS columns

Name	Type	Reference	Description
objoid	oid	Any OID column	OID of the object that this security label pertains to.
classoid	oid	**oid** in PG_CLASS	OID of the system catalog where the object to which the security label belongs is located.
objsubid	integer	-	Column number for the security label on a table column (**objoid** and **classoid** refer to the table itself). The value is **0** for all other object types.
objtype	text	-	Type of the object to which the label belongs, in text format. Example: • **table**: table type • **column**: column type
objnamespace	oid	**oid** in PG_NAMESPACE	OID of the namespace for the object, if applicable; otherwise, **NULL**.
objname	text	-	Name of the object to which the label belongs, in text format.
provider	text	**provider** in PG_SECLABEL	Provider of the label.
label	text	**label** in PG_SECLABEL	Security label name.

12.3.7.19 PG_SHADOW

PG_SHADOW displays the attributes of all roles marked with rolcanlogin in **PG_AUTHID**. Only the system administrator can access this system view.

The information in this view is basically the same as that in **PG_USER**. The difference is that in **PG_USER**, passwords are sensitive and displayed as ********.

Table 12-215 PG_SHADOW columns

Name	Type	Reference	Description
usename	name	**rolname** in **PG_AUTHID**	Username.
usesysid	oid	**oid** in **PG_AUTHID**	ID of this user.
usecreatedb	Boolean	-	Specifies whether a user has the permissions to create databases. • **t** (true): yes. • **f** (false): no.
usesuper	Boolean	-	Specifies whether the user is a system administrator. • **t** (true): yes. • **f** (false): no.
usecatupd	Boolean	-	Specifies whether the user can update a view. Even the system administrator cannot do this unless this column is **true**.
userepl	Boolean	-	Specifies whether the user has the permissions to duplicate data streams. • **t** (true): yes. • **f** (false): no.
passwd	text	**rolpassword** in **PG_AUTHID**	Password ciphertext. If there is no password, the value is **NULL**.
valbegin	timestamp with time zone	-	Start time for account validity (**NULL** if the start time is not specified).
valuntil	timestamp with time zone	-	End time for account validity (**NULL** if the end time is not specified).
respool	name	-	Resource pool where the user is in.
parent	oid	-	Parent user OID.

Name	Type	Reference	Description
spacelimit	text	-	Storage space of the permanent table, in KB.
useconfig	text[]	**setconfig** in PG_DB_ROLE_SETTING	Default value of runtime configuration items.
tempspacelimit	text	-	Storage space of the temporary table, in KB.
spillspacelimit	text	-	Operator disk spill space, in KB.
usemonitoradmin	Boolean	-	Specifies whether the user is a monitor administrator. • **t** (true): yes. • **f** (false): no.
useoperatoradmin	Boolean	-	Specifies whether the user is an O&M administrator. • **t** (true): yes. • **f** (false): no.
usepolicyadmin	Boolean	-	Specifies whether the user is a security policy administrator. • **t** (true): yes. • **f** (false): no.

12.3.7.20 PG_USER

PG_USER displays information about database users. By default, only the initial user and users with the sysadmin attribute can view the information. Other users can view the information only after being granted with permissions.

Table 12-216 PG_USER columns

Name	Type	Description
usename	name	Username.
usesysid	oid	ID of this user.
usecreatedb	Boolean	Specifies whether a user has the permissions to create databases. • **t** (true): yes. • **f** (false): no.

Name	Type	Description
usesuper	Boolean	Specifies whether the user is the initial system administrator with the highest permissions. • **t** (true): yes. • **f** (false): no.
usecatupd	Boolean	Specifies whether the user can directly update system catalogs. Only the initial system administrator whose **usesysid** is **10** has this permission. This permission is unavailable for other users. • **t** (true): yes. • **f** (false): no.
userepl	Boolean	Specifies whether the user has the permissions to duplicate data streams. • **t** (true): yes. • **f** (false): no.
passwd	text	Encrypted user password. The value is displayed as ********.
valbegin	timestamp with time zone	Start time for account validity (**NULL** if the start time is not specified).
valuntil	timestamp with time zone	End time for account validity (**NULL** if the end time is not specified).
respool	name	Resource pool where the user is in.
parent	oid	Parent user OID.
spacelimit	text	Storage space of the permanent table, in KB.
useconfig	text[]	Default value of runtime configuration items. For details, see **setconfig** in **PG_DB_ROLE_SETTING**.
nodegroup	name	Name of the logical database associated with the user. If the user does not manage the logical database, this column is left blank.
tempspacelimit	text	Storage space of the temporary table, in KB.
spillspacelimit	text	Operator disk spill space, in KB.

Name	Type	Description
usemonitoradmin	Boolean	Specifies whether the user is a monitor administrator. • **t** (true): yes. • **f** (false): no.
useoperatoradmin	Boolean	Specifies whether the user is an O&M administrator. • **t** (true): yes. • **f** (false): no.
usepolicyadmin	Boolean	Specifies whether the user is a security policy administrator. • **t** (true): yes. • **f** (false): no.

12.3.7.21 PG_USER_MAPPINGS

PG_USER_MAPPINGS displays user mapping information.

This is essentially a publicly readable view of **PG_USER_MAPPING** that leaves out the options column if the user has no rights to use the system catalog and query the view. Common users must be authorized to access this view.

Table 12-217 PG_USER_MAPPINGS columns

Name	Type	Reference	Description
umid	oid	**oid** in **PG_USER_MAPPING**	OID of the user mapping.
srvid	oid	**oid** in **PG_FOREIGN_SERVER**	OID of the foreign server that contains the mapping.
srvname	name	**srvname** in **PG_FOREIGN_SERVER**	Name of the foreign server.
umuser	oid	**oid** in **PG_AUTHID**	OID of the local role being mapped (**0** if the user mapping is public).
usename	name	-	Name of the local user to be mapped.
umoptions	text[]	-	User mapping specific options. If the current user is the owner of the foreign server, the value is **keyword=value strings**. Otherwise, the value is **NULL**.

12.3.7.22 ROLE_ROLE_PRIVS

ROLE_ROLE_PRIVS displays roles granted to other roles and provides only information about the roles that the user has access to. By default, it is accessible to all users. This view exists in both PG_CATALOG and SYS schema.

Table 12-218 ROLE_ROLE_PRIVS columns

Name	Type	Description
role	character varying(128)	Role name.
granted_role	character varying(128)	Role to be granted.
admin_option	character varying(3)	Specifies whether the grant contains the **ADMIN** option. • **YES** • **NO**
common	character varying(3)	Not supported. Its value is **NULL**.
inherited	character varying(3)	Not supported. Its value is **NULL**.

12.3.7.23 ROLE_SYS_PRIVS

ROLE_SYS_PRIVS displays information about system privileges granted to roles (only roles accessible to the user are displayed). By default, it is accessible to all users. This view exists in both PG_CATALOG and SYS schema.

Table 12-219 ROLE_SYS_PRIVS columns

Name	Type	Description
role	character varying(128)	Role name.

Name	Type	Description
privilege	character varying(40)	System permissions or ANY permissions of the user. • System permissions include rolsuper, rolinherit, rolcreaterole, rolcreatedb, rolcatupdate, rolcanlogin, rolreplication, rolauditadmin, rolsystemadmin, roluseft, rolmonitoradmin, roloperatoradmin, and rolpolicyadmin. • For details about the value of the ANY permissions, see Table 7-223.
admin_option	character varying(3)	Specifies whether the grant contains the **ADMIN** option. • **YES** • **NO**
common	character varying(3)	Not supported. Its value is **NULL**.
inherited	character varying(3)	Not supported. Its value is **NULL**.

12.3.7.24 ROLE_TAB_PRIVS

ROLE_TAB_PRIVS displays information about object permissions granted to roles (only roles accessible to the user are displayed). It is accessible to all users. This view exists in the PG_CATALOG and SYS schemas.

Table 12-220 ROLE_TAB_PRIVS columns

Name	Type	Description
role	character varying(128)	Role name.
owner	character varying(128)	Object owner.
table_name	character varying(128)	Object name. Object types include tables, packages, indexes, and sequences.
column_name	character varying(128)	Not supported. Set it to **NULL**.

Name	Type	Description
privilege	character varying(40)	Permissions on an object, including USAGE, UPDATE, DELETE, INSERT, CONNECT, SELECT, and EXECUTE.
grantable	character varying(3)	Specifies whether the grant contains the **GRANT** option. • **YES** • **NO**
common	character varying(3)	Not supported. Set it to **NULL**.
inherited	character varying(3)	Not supported. Set it to **NULL**.

12.3.8 Dynamic Data Masking

12.3.8.1 GS_MASKING

GS_MASKING displays all configured dynamic masking policies. Only the users with system administrator or security policy administrator permission can access this view.

Table 12-221 GS_MASKING columns

Name	Type	Description
polname	name	Name of the masking policy
polenabled	Boolean	Specifies whether to enable the masking policy.
maskaction	name	Masking function
labelname	name	Name of the label to which the masking function applies.
masking_object	text	Masking database resource object
filter_name	text	Logical expression of a filter criterion

12.3.9 Transparent Encryption

12.3.9.1 PG_TDE_INFO

PG_TDE_INFO displays encryption information of the entire database.

Table 12-222 PG_TDE_INFO columns

Name	Type	Description
is_encrypt	Boolean	Specifies whether to encrypt the database. • **f**: The database is not encrypted. • **t**: The database is encrypted.
g_tde_algo	text	Encryption algorithm. • **SM4-CTR-128** • **AES-CTR-128**
remain	text	Reserved column.

12.3.10 DATABASE LINK

12.3.10.1 GS_DB_LINKS

GS_DB_LINKS displays information about DATABASE LINK objects. You can view information about your own DATABASE LINK objects and DATABASE LINK objects at the PUBLIC level.

Table 12-223 GS_DB_LINKS columns

Name	Type	Description
dblinkid	oid	OID of the current DATABASE LINK object.
dlname	name	Name of the current DATABASE LINK object.
dlowner	oid	Owner ID of the current DATABASE LINK object. If the object owner is **public**, the value is **0**.
dlownername	name	Name of the owner of the current DATABASE LINK object.
options	text[]	Connection information of the current DATABASE LINK object. The value is a character string in the "keyword=value" format.
useroptions	text	Information of the remote end user to which the current DATABASE LINK object connects.
heterogeneous	text	Not supported. Its value is **NULL**.

Name	Type	Description
protocol	text	Not supported. Its value is **NULL**.
opencursors	text	Not supported. Its value is **NULL**.
intransaction	Boolean	Specifies whether the current DATABASE LINK object exists in a transaction.
updatasent	Boolean	Specifies whether the current DATABASE LINK object uses statements for updating data.

12.3.10.2 V$DBLINK

V$DBLINK displays information about database links. You can view information about your own database links and database links at the PUBLIC level. By default, only administrators can view this view.

NOTE

The data in this view is obtained from **GS_DB_LINKS**. This view records only the information about the links that have been used or are being used in the current session. For details about database links, see **DATABASE LINK**.

Table 12-224 V$DBLINK columns

Name	Type	Description
db_link	character varying(128)	Name of the current database link.
owner_id	numeric	Owner ID of the current database link. If the owner of the database link is **PUBLIC**, the value of **owner_id** is **0**.
logged_on	character varying(3)	Not supported. Set it to **NULL**.
heterogeneous	character varying(3)	Not supported. Set it to **NULL**.
protocol	character varying(6)	Not supported. Set it to **NULL**.

Name	Type	Description
open_cursors	numeric	Not supported. Set it to **NULL**.
in_transaction	character varying(3)	Determines whether the current database link exists in a transaction. • **YES** • **NO**
update_sent	character varying(3)	Determines whether the current database link uses statements for updating data. • **YES** • **NO**
commit_point_strength	numeric	Not supported. Set it to **NULL**.
con_id	numeric	Not supported. Set it to **NULL**.

12.3.11 Materialized Views

12.3.11.1 GS_MATVIEWS

GS_MATVIEWS displays information about each materialized view in the database.

Table 12-225 GS_MATVIEWS columns

Name	Type	Reference	Description
schemaname	name	PG_NAMESPACE.nspname	Name of the schema of a materialized view.
matviewname	name	PG_CLASS.relname	Name of a materialized view.
matviewowner	name	PG_AUTHID.rolname	Owner of a materialized view.
tablespace	name	PG_TABLESPACE.spcname	Tablespace name of a materialized view. If the default tablespace of the database is used, the value is null.
hasindexes	Boolean	-	This column is true if a materialized view has (or has recently had) any indexes.

Name	Type	Reference	Description
definition	text	-	Definition of a materialized view (a reconstructed SELECT query)

12.3.12 Other System Views

12.3.12.1 ADM_ARGUMENTS

ADM_ARGUMENTS displays parameter information of all stored procedures or functions. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG_CATALOG and SYS schema.

Table 12-226 ADM_ARGUMENTS columns

Name	Type	Description
owner	character varying(128)	Owner of a function or stored procedure.
object_name	character varying(128)	Name of a function or stored procedure.
package_name	character varying(128)	Package name.
object_id	oid	OID of a function or stored procedure.
overload	character varying(40)	*n*th overloaded function of the name.
subprogram_id	numeric	Location of a function or stored procedure in a package.
argument_name	character varying(128)	Parameter name.
position	numeric	Position of the parameter in the parameter list. The value is **0** for the return value of the function by default.
sequence	numeric	Sequence of a parameter, which starts from 1, with the return type before all parameters.

Name	Type	Description
data_level	numeric	Nesting depth of parameters of the composite type. The value is fixed at **0** because only one line is displayed for each parameter.
data_type	character varying(30)	Data type of a parameter.
defaulted	character varying(1)	Specifies whether a parameter has a default value: ● **Y**: yes. ● **N**: no.
default_value	text	Not supported. Its value is **NULL**.
default_length	numeric	Not supported. Its value is **NULL**.
in_out	character varying(9)	Input and output attributes of a parameter: ● **IN**: input parameter. ● **OUT**: output parameter. ● **IN_OUT**: input and output parameters. ● **VARIADIC**: VARIADIC parameter.
data_length	numeric	Not supported. Its value is **NULL**.
data_precision	numeric	Not supported. Its value is **NULL**.
data_scale	numeric	Not supported. Its value is **NULL**.
radix	numeric	Radix of a number, which is **10** when the data type is smallint, integer, bigint, numeric, or float. For other data types, set this column to **NULL**.
character_set_name	character varying(44)	Not supported. Its value is **NULL**.

Name	Type	Description
type_owner	character varying(128)	Owner of the data type.
type_name	character varying(128)	Parameter type name. Only the customized type is displayed.
type_subname	character varying(128)	Not supported. Its value is **NULL**.
type_link	character varying(128)	Not supported. Its value is **NULL**.
type_object_type	character varying(7)	Types of types of the **type_owner**, **type_name**, and **type_subname** columns: • **TABLE**: The parameter is of the table type. • **VIEW**: The parameter is of the view type. • For other data types, the value is **NULL**.
pls_type	character varying(128)	Name of the PL/SQL type for parameters of the number type. Otherwise, this column is empty.
char_length	numeric	Not supported. Its value is **NULL**.
char_used	character varying(1)	Not supported. Set it to **B** if the data type is varchar, nvarchar2, bpchar, or char. For other data types, set it to **NULL**.
origin_con_id	character varying(256)	Not supported. Its value is **0**.

12.3.12.2 ADM_COL_COMMENTS

ADM_COL_COMMENTS displays information about table column comments in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-227 ADM_COL_COMMENTS columns

Name	Type	Description
owner	character varying(128)	Table owner.
table_name	character varying(128)	Table name.
column_name	character varying(128)	Column name.
comments	text	Comments.
origin_con_id	numeric	Not supported. Set it to **0**.
schema	character varying(64)	Name of the namespace to which the column belongs.

12.3.12.3 ADM_COLL_TYPES

ADM_COLL_TYPES displays information about all collection types. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-228 ADM_COLL_TYPES columns

Name	Type	Description
owner	character varying(128)	Owner of a collection.
type_name	character varying(128)	Name of a collection.
coll_type	character varying(128)	Description of a collection.
upper_bound	numeric	Not supported. Set it to **NULL**.
elem_type_mod	character varying(7)	Type modifier of an element.
elem_type_owner	character varying(128)	Owner of the element type on which the collection is based. This parameter is mainly used for user-defined types.
elem_type_name	character varying(128)	Name of the data type or user-defined type on which the collection is based.
length	numeric	Not supported. Set it to **NULL**.
precision	numeric	Not supported. Set it to **NULL**.
scale	numeric	Not supported. Set it to **NULL**.

Name	Type	Description
character_set_name	character varying(44)	Not supported. Set it to **NULL**.
elem_storage	character varying(7)	Not supported. Set it to **NULL**.
nulls_stored	character varying(3)	Not supported. Set it to **NULL**.
char_used	character varying(1)	Not supported. Set it to **NULL**.

12.3.12.4 ADM_CONS_COLUMNS

ADM_CONS_COLUMNS displays information about constraint columns in database tables. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-229 ADM_CONS_COLUMNS columns

Name	Type	Description
owner	character varying(64)	Constraint creator.
constraint_name	character varying(64)	Constraint name.
table_name	character varying(64)	Name of a constraint-related table.
column_name	character varying(64)	Name of a constraint-related column.
position	smallint	Position of a column in a table.

12.3.12.5 ADM_CONSTRAINTS

ADM_CONSTRAINTS displays information about table constraints in database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-230 ADM_CONSTRAINTS columns

Name	Type	Description
owner	character varying(64)	Constraint creator.

Name	Type	Description
constraint_name	character varying(64)	Constraint name.
constraint_type	text	Constraint type. • **c**: check constraint • **f**: foreign key constraint • **p**: primary key constraint • **u**: unique constraint
table_name	character varying(64)	Name of a constraint-related table.
index_owner	character varying(64)	Owner of a constraint-related index (only for the unique constraint and primary key constraint).
index_name	character varying(64)	Name of a constraint-related index (only for the unique constraint and primary key constraint).
status	character varying(8)	Constraint status
generated	character varying(14)	Not supported. Set it to **NULL**.
search_condition	text	Not supported. Set it to **NULL**.
search_condition_vc	character varying(4000)	Not supported. Set it to **NULL**.
r_owner	character varying(128)	Not supported. Set it to **NULL**.
r_constraint_name	character varying(128)	Not supported. Set it to **NULL**.
delete_rule	character varying(9)	Not supported. Set it to **NULL**.
con_deferrable	character varying(14)	Not supported. Set it to **NULL**.
deferred	character varying(9)	Not supported. Set it to **NULL**.
validated	character varying(13)	Not supported. Set it to **NULL**.
bad	character varying(3)	Not supported. Set it to **NULL**.
rely	character varying(4)	Not supported. Set it to **NULL**.
last_change	timestamp(0) without time zone	Not supported. Set it to **NULL**.
invalid	character varying(7)	Not supported. Set it to **NULL**.
view_related	character varying(14)	Not supported. Set it to **NULL**.
origin_con_id	character varying(256)	Not supported. Set it to **NULL**.

12.3.12.6 ADM_DATA_FILES

ADM_DATA_FILES displays the description of database files. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-231 ADM_DATA_FILES columns

Name	Type	Description
tablespace_name	name	Name of the tablespace to which a file belongs
bytes	double precision	Length of a file, in bytes

12.3.12.7 ADM_DEPENDENCIES

ADM_DEPENDENCIES displays the dependency relationships between types, tables, views, stored procedures, functions, and triggers in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-232 ADM_DEPENDENCIES columns

Name	Type	Description
owner	name	Object owner.
name	name	Object name.
type	character varying(18)	Object class.
referenced_owner	name	Owner of the referenced object.
referenced_name	name	Name of the referenced object.
referenced_type	character varying(18)	Type of the referenced object.
referenced_link_name	character varying(128)	Not supported. Set it to **NULL**.
dependency_type	character varying(4)	Not supported. Set it to **NULL**.

12.3.12.8 ADM_DIRECTORIES

ADM_DIRECTORIES stores information about all directory objects in the database. By default, only the system administrator can access this view. Common users can

access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-233 ADM_DIRECTORIES columns

Name	Type	Description
owner	oid	Directory owner
directory_name	name	Directory name
directory_path	text	Operating system path name of a directory
origin_con_id	character varying(256)	Not supported. Set it to **NULL**.

12.3.12.9 ADM_HIST_SNAPSHOT

ADM_HIST_SNAPSHOT records the WDR snapshot data stored in the current system. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG_CATALOG and SYS schema. This view can be accessed only when the GUC parameter **enable_wdr_snapshot** is set to **on**. To access the PG_CATALOG.ADM_HIST_SNAPSHOT and SYS.ADM_HIST_SNAPSHOT views, you must have the permission to access the **snapshot schema**, **snapshot**, and **tables_snap_timestamp** tables.

Table 12-234 ADM_HIST_SNAPSHOT columns

Name	Type	Description
snap_id	bigint	Unique snapshot ID.
dbid	oid	Database ID of the snapshot.
instance_number	oid	Not supported. Its value is the same as that of **dbid**.
startup_time	timestamp(3) without time zone	Instance start time.
begin_interval_time	timestamp without time zone	Start time of the snapshot interval (the end time of the last snapshot).
end_interval_time	timestamp without time zone	End time of the snapshot interval. Actual time when the snapshot is taken (that is, the end time of the snapshot).
flush_elapsed	interval	Amount of time to perform the snapshot.

Name	Type	Description
snap_level	numeric	Not supported. Its value is **NULL**.
error_count	numeric	Not supported. Its value is **NULL**.
snap_flag	numeric	Not supported. Its value is **NULL**.
snap_timezone	interval day to second(0)	Time offset between the snapshot time zone and the coordinated universal time (UTC).
begin_interval_time_tz	timestamp with time zone	Start time of the snapshot interval with the time zone (end time of the last snapshot).
end_interval_time_tz	timestamp with time zone	End time of the snapshot interval. Actual time when the snapshot is taken (that is, the end time of the snapshot), with the time zone.
con_id	numeric	Not supported. Its value is **0**.

12.3.12.10 ADM_HIST_SQL_PLAN

ADM_HIST_SQL_PLAN displays plan information collected by the current user by running the **EXPLAIN PLAN** statement. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-235 ADM_HIST_SQL_PLAN columns

Name	Type	Description
dbid	text	Database ID
sql_id	character varying(30)	Session that inserts the data. Its value consists of a service thread start timestamp and a service thread ID. Values are constrained by **NOT NULL**.
plan_hash_value	bigint	Query ID
id	integer	Number assigned to each step in the execution plan
operation	character varying(30)	Operation description
options	character varying(255)	Operation option
object_node	character varying(128)	Not supported. Set it to **NULL**.
object#	numeric	Not supported. Set it to **NULL**.

Name	Type	Description
object_owner	name	Object number of a table or an index
object_name	name	Name of the operated object. It is defined by users.
object_alias	character varying(261)	Not supported. Set it to **NULL**.
object_type	character varying(30)	Object type.
optimizer	character varying(20)	Not supported. Set it to **NULL**.
parent_id	numeric	Not supported. Set it to **NULL**.
depth	numeric	Not supported. Set it to **NULL**.
position	numeric	Not supported. Set it to **NULL**.
search_columns	numeric	Not supported. Set it to **NULL**.
cost	double precision	Execution cost estimated by the optimizer for an operator
cardinality	double precision	Cardinality estimated by the optimizer for an operator to access table records.
bytes	numeric	Not supported. Set it to **NULL**.
other_tag	character varying(35)	Not supported. Set it to **NULL**.
partition_start	character varying(64)	Not supported. Set it to **NULL**.
partition_stop	character varying(64)	Not supported. Set it to **NULL**.
partition_id	numeric	Not supported. Set it to **NULL**.
other	character varying(4000)	Not supported. Set it to **NULL**.
distribution	character varying(20)	Not supported. Set it to **NULL**.
cpu_cost	numeric	Not supported. Set it to **NULL**.
io_cost	numeric	Not supported. Set it to **NULL**.
temp_space	numeric	Not supported. Set it to **NULL**.

Name	Type	Description
access_predicates	character varying(4000)	Not supported. Set it to **NULL**.
filter_predicates	character varying(4000)	Not supported. Set it to **NULL**.
projection	character varying(4000)	Returned column information
time	numeric	Not supported. Set it to **NULL**.
qblock_name	character varying(128)	Not supported. Set it to **NULL**.
remarks	character varying(4000)	Not supported. Set it to **NULL**.
timestamp	timestamp(0) without time zone	Not supported. Set it to **NULL**.
other_xml	clob	Not supported. Set it to **NULL**.
con_dbid	text	Database ID of a container, which is currently set to a value the same as that of **dbid**.
con_id	numeric	Container ID. Containers are not supported and this field is set to **0**.

12.3.12.11 ADM_HIST_SQLSTAT

ADM_HIST_SQLSTAT displays information about statements executed on the current node. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

After the WDR snapshot function is enabled (that is, the GUC parameter **enable_wdr_snapshot** is set to **on**), users can view the data in this view.

Table 12-236 ADM_HIST_SQLSTAT columns

Name	Type	Description
instance_number	integer	Instance ID of a snapshot.
plan_hash_value	integer	ID of the normalized SQL statement.

Name	Type	Description
module	integer	Name of the module that is executing when the SQL statement is first parsed.
apwait_delta	integer	Delta value of the application wait time.
sql_id	bigint	Query ID.
snap_id	bigint	Unique snapshot ID.
elapsed_time_delta	bigint	Valid DB time, which is accumulated if multiple threads are involved (unit: μs).
cpu_time_delta	bigint	CPU time (unit: μs).
executions_delta	bigint	Increment in the number of executions that have occurred on this object since it was brought into the cache.
iowait_delta	bigint	I/O time (unit: μs).
rows_processed_delta	bigint	Number of rows in the result set returned by the SELECT statement.
parsing_schema_name	character varying	Not supported. Its value is **NULL**.
disk_reads_delta	bigint	Not supported. Its value is **NULL**.
buffer_reads_delta	bigint	Not supported. Its value is **NULL**.
clwait_delta	bigint	Not supported. Its value is **NULL**.

12.3.12.12 ADM_HIST_SQLTEXT

ADM_HIST_SQLTEXT displays information about statements executed on the current node. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

After the WDR snapshot function is enabled (that is, the GUC parameter **enable_wdr_snapshot** is set to **on**), users can view the data in this view.

Table 12-237 ADM_HIST_SQLTEXT columns

Name	Type	Description
dbid	integer	Database ID
sql_id	bigint	Query ID

Name	Type	Description
sql_text	clob	Text corresponding to the query
command_type	integer	Not supported. Set it to **0**.
con_dbid	integer	Database ID of a container, which is currently set to a value the same as that of **dbid**.
con_id	integer	Container ID. Containers are not supported and this field is set to **0**.

12.3.12.13 ADM_IND_COLUMNS

ADM_IND_COLUMNS displays information about index columns in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG_CATALOG and SYS schema.

Table 12-238 ADM_IND_COLUMNS columns

Name	Type	Description
index_owner	character varying(64)	Index owner.
index_name	character varying(64)	Index name.
table_owner	character varying(64)	Table owner.
table_name	character varying(64)	Table name.
column_name	name	Column name.
column_position	smallint	Position of a column in an index.
column_length	numeric	Length of the column. For the variable-length type, its value is **NULL**.
char_length	numeric	Maximum length of a column, in bytes.
descend	character varying(4)	Columns sorted in descending (**DESC**) or ascending (**ASC**) order.
collated_column_id	numeric	Not supported. Its value is **NULL**.

12.3.12.14 ADM_IND_EXPRESSIONS

ADM_IND_EXPRESSIONS displays information about expression indexes in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-239 ADM_IND_EXPRESSIONS columns

Name	Type	Description
table_owner	character varying(64)	Table owner.
table_name	character varying(64)	Table name.
index_owner	character varying(64)	Index owner.
index_name	character varying(64)	Index name.
column_expression	text	Function-based index expression of a specified column.
column_position	smallint	Position of a column in an index.

12.3.12.15 ADM_INDEXES

ADM_INDEXES displays all indexes in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG_CATALOG and SYS schema.

Table 12-240 ADM_INDEXES columns

Name	Type	Description
owner	character varying(64)	Index owner.
index_name	character varying(64)	Index name.
table_name	character varying(64)	Name of a table corresponding to an index.
uniqueness	text	Specifies whether the index is unique. • **UNIQUE**: unique index. • **NONUNIQUE**: non-unique index.

Name	Type	Description
partitioned	character(3)	Specifies whether the index has the property of partitioned tables. • **Yes**: The index has the property of a partitioned table. • **No**: The index does not have the property of a partitioned table.
generated	character varying(1)	Specifies whether the name of the index is generated by the system. • **y**: The index name is generated by the system. • **n**: The index name is not generated by the system.
index_type	character varying(27)	Index type. • **NORMAL**: Index attributes are simple references, and the expression tree is empty. • **FUNCTION-BASED NORMAL**: Expression trees are used for index attributes that are not simple column references.
table_owner	character varying(128)	Owner of an index object.
table_type	character(11)	Type of an index object. • **TABLE**: The index object is of the table type.
tablespace_name	character varying(30)	Name of a tablespace that contains the index.

Name	Type	Description
status	character varying(8)	Status of a non-partitioned index. • **VALID**: Non-partitioned indexes can be used for query. • **UNUSABLE**: The non-partitioned index is unavailable. • **N/A**: The index has the property of a partitioned table.
compression	character varying(13)	Not supported. Its value is **NULL**.
prefix_length	numeric	Not supported. Its value is **NULL**.
ini_trans	numeric	Not supported. Its value is **NULL**.
max_trans	numeric	Not supported. Its value is **NULL**.
initial_extent	numeric	Not supported. Its value is **NULL**.
next_extent	numeric	Not supported. Its value is **NULL**.
min_extents	numeric	Not supported. Its value is **NULL**.
max_extents	numeric	Not supported. Its value is **NULL**.
pct_increase	numeric	Not supported. Its value is **NULL**.
pct_threshold	numeric	Not supported. Its value is **NULL**.
include_column	numeric	Not supported. Its value is **NULL**.
freelists	numeric	Not supported. Its value is **NULL**.
freelist_groups	numeric	Not supported. Its value is **NULL**.
pct_free	numeric	Not supported. Its value is **NULL**.

Name	Type	Description
logging	character varying(3)	Not supported. Its value is **NULL**.
blevel	numeric	Not supported. Its value is **NULL**.
leaf_blocks	numeric	Not supported. Its value is **NULL**.
distinct_keys	numeric	Not supported. Its value is **NULL**.
avg_leaf_blocks_per_key	numeric	Not supported. Its value is **NULL**.
avg_data_blocks_per_key	numeric	Not supported. Its value is **NULL**.
clustering_factor	numeric	Not supported. Its value is **NULL**.
num_rows	numeric	Not supported. Its value is **NULL**.
sample_size	numeric	Not supported. Its value is **NULL**.
last_analyzed	timestamp(0) without time zone	Not supported. Its value is **NULL**.
degree	character varying(40)	Not supported. Its value is **NULL**.
instances	character varying(40)	Not supported. Its value is **NULL**.
temporary	character varying(1)	Not supported. Its value is **NULL**.
secondary	character varying(1)	Not supported. Its value is **NULL**.
buffer_pool	character varying(7)	Not supported. Its value is **NULL**.
flash_cache	character varying(7)	Not supported. Its value is **NULL**.
cell_flash_cache	character varying(7)	Not supported. Its value is **NULL**.
user_stats	character varying(3)	Not supported. Its value is **NULL**.
duration	character varying(15)	Not supported. Its value is **NULL**.

Name	Type	Description
pct_direct_access	numeric	Not supported. Its value is **NULL**.
ityp_owner	character varying(128)	Not supported. Its value is **NULL**.
ityp_name	character varying(128)	Not supported. Its value is **NULL**.
parameters	character varying(1000)	Not supported. Its value is **NULL**.
global_stats	character varying(3)	Not supported. Its value is **NULL**.
domidx_status	character varying(12)	Not supported. Its value is **NULL**.
domidx_opstatus	character varying(6)	Not supported. Its value is **NULL**.
funcidx_status	character varying(8)	Not supported. Its value is **NULL**.
join_index	character varying(3)	Not supported. Its value is **NULL**.
iot_redundant_pkey_elim	character varying(3)	Not supported. Its value is **NULL**.
dropped	character varying(3)	Not supported. Its value is **NULL**.
visibility	character varying(9)	Specifies whether the index is visible to the optimizer. • **VISIBLE**: The index is visible to the optimizer. • **INVISIBLE**: The index is invisible to the optimizer.
domidx_management	character varying(14)	Not supported. Its value is **NULL**.
segment_created	character varying(3)	Not supported. Its value is **NULL**.
orphaned_entries	character varying(3)	Not supported. Its value is **NULL**.
indexing	character varying(7)	Not supported. Its value is **NULL**.

Name	Type	Description
auto	character varying(3)	Not supported. Its value is **NULL**.

12.3.12.16 ADM_OBJECTS

ADM_OBJECTS displays all database objects in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-241 ADM_OBJECTS columns

Name	Type	Description
owner	name	Object owner.
object_name	name	Object name.
object_id	oid	Object OID.
object_type	name	Object class. For example, table, schema, and index.
namespace	oid	Namespace containing an object
temporary	character(1)	Specifies whether an object is a temporary object.
status	character varying(7)	Object status.
subobject_name	name	Subobject name of an object.
generated	character(1)	Specifies whether an object name is generated by the system.
created	timestamp with time zone	Creation time of an object.
last_ddl_time	timestamp with time zone	Last modification time of an object.
default_collation	character varying(100)	Default collation of objects.
data_object_id	numeric	Not supported. Set it to **NULL**.
timestamp	character varying(19)	Not supported. Set it to **NULL**.

Name	Type	Description
secondary	character varying(1)	Not supported. Set it to **NULL**.
edition_name	character varying(128)	Not supported. Set it to **NULL**.
sharing	character varying(18)	Not supported. Set it to **NULL**.
editionable	character varying(1)	Not supported. Set it to **NULL**.
oracle_maintained	character varying(1)	Not supported. Set it to **NULL**.
application	character varying(1)	Not supported. Set it to **NULL**.
duplicated	character varying(1)	Not supported. Set it to **NULL**.
sharded	character varying(1)	Not supported. Set it to **NULL**.
created_appid	numeric	Not supported. Set it to **NULL**.
modified_appid	numeric	Not supported. Set it to **NULL**.
created_vsnid	numeric	Not supported. Set it to **NULL**.
modified_vsnid	numeric	Not supported. Set it to **NULL**.

NOTICE

For details about the value ranges of **created** and **last_ddl_time**, see **PG_OBJECT**.

12.3.12.17 ADM_PROCEDURES

ADM_PROCEDURES displays information about all stored procedures, functions, and triggers in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-242 ADM_PROCEDURES columns

Name	Type	Description
owner	character varying(64)	Owner of a stored procedure, function, trigger, or package

Name	Type	Description
object_name	character varying(64)	Name of a stored procedure, function, or trigger. This column will be a package name if the object is a function in a package or a stored procedure.
procedure_name	character varying(128)	This column will be the name of a function or stored procedure in a package if **object_name** is a package name. Otherwise, the column is empty.
object_id	oid	OID of a stored procedure, function, trigger, or package
subprogram_id	numeric	Location of a function or stored procedure in a package.
overload	character varying(40)	Mth overloaded function.
object_type	character varying(13)	Object class.
aggregate	character varying(3)	Specifies whether the function is an aggregate function: • **YES** • **NO**
pipelined	character varying(3)	Not supported. Set it to **NO**.
impltypeowner	character varying(128)	Owner of an implementation type
impltypename	character varying(128)	Name of an implementation type
parallel	character varying(3)	Not supported. Set it to **NO**.
interface	character varying(3)	Not supported. Set it to **NO**.
deterministic	character varying(3)	Not supported. Set it to **NO**.

Name	Type	Description
authid	character varying(12)	Whether to use the creator permission or caller permission: • **DEFINER**: The creator permission is used. • **CURRENT_USER**: The caller permission is used. This column conflicts with reserved keywords. Therefore, add the view name when calling this column.
result_cache	character varying(3)	Not supported. Set it to **NULL**.
origin_con_id	character varying(256)	Not supported. Set it to **0**.
polymorphic	character varying(5)	Not supported. Set it to **NULL**.
argument_number	smallint	Number of input parameters in a stored procedure

12.3.12.18 ADM_RECYCLEBIN

ADM_RECYCLEBIN displays information about all recycle bins. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG_CATALOG and SYS schema.

Table 12-243 ADM_RECYCLEBIN columns

Name	Type	Description
owner	character varying(128)	Original owner name of the object.
object_name	character varying(128)	New name of an object.
original_name	character varying(128)	Original name of an object.

Name	Type	Description
operation	character varying(18)	Operation performed on an object. The options are as follows: ● **DROP**: The object is deleted (the object is no longer required). ● **TRUNCATE**: The object is truncated.
type	character varying(25)	Object type.
ts_name	character varying(30)	Name of the tablespace to which the object belongs.
createtime	character varying(19)	Timestamp when an object is created.
droptime	character varying(19)	Timestamp when an object is deleted.
dropscn	numeric	System change number (SCN) of the transaction that moves an object to the recycle bin.
partition_name	character varying(128)	Name of a deleted partition.
can_undrop	character varying(3)	Specifies whether to flash back an object.
can_purge	character varying(3)	Specifies whether to purge an object.
related	numeric	ID of the parent object.
base_object	oid	ID of a base object.
purge_object	oid	ID of the purged object.
space	numeric	Number of blocks used by an object.

12.3.12.19 ADM_SCHEDULER_JOB_ARGS

ADM_SCHEDULER_JOB_ARG displays parameters related to all jobs in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-244 ADM_SCHEDULER_JOB_ARGS columns

Name	Type	Description
owner	character varying(128)	Owner of the job to which the parameter belongs.
job_name	character varying(128)	Name of the job to which the parameter belongs.
argument_name	character varying(128)	Parameter name.
argument_position	numeric	Position of the parameter in the parameter list.
argument_type	character varying(257)	Data type of a parameter.
value	character varying(4000)	Parameter value.
anydata_value	character varying(4000)	Not supported. Set it to **NULL**.
out_argument	character varying(5)	Reserved column. Set it to **NULL**.

12.3.12.20 ADM_SCHEDULER_JOBS

ADM_SCHEDULER_JOBS displays information about all DBE_SCHEDULER scheduled jobs in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG_CATALOG and SYS schema.

Table 12-245 ADM_SCHEDULER_JOBS columns

Name	Type	Description
owner	name	Owner of a scheduled job.
job_name	text	Name of a scheduled job.
job_subname	character varying(128)	Not supported. Its value is **NULL**.
job_style	text	Action mode of a scheduled job. It is specified during creation. Its value can only be **"REGULAR"**. If this parameter is not specified, the value is **NULL**.
job_creator	name	Creator of a scheduled job.

Name	Type	Description
client_id	character varying(65)	Not supported. Its value is **NULL**.
global_uid	character varying(33)	Not supported. Its value is **NULL**.
program_owner	character varying(4000)	Owner of a program referenced by a scheduled job.
program_name	text	Name of the program referenced by a scheduled job.
job_type	character varying(16)	Inline program type of a scheduled job. The options are as follows: • **PLSQL_BLOCK**: anonymous block of a stored procedure. • **STORED_PROCEDURE**: stored procedure that is saved. • **EXTERNAL_SCRIPT**: external script.
job_action	text	Program content of a scheduled job.
number_of_arguments	text	Number of parameters of a scheduled job.
schedule_owner	character varying(4000)	Not supported. Its value is **NULL**.
schedule_name	text	Name of the schedule referenced by a scheduled job.
schedule_type	character varying(12)	Not supported. Its value is **NULL**.
start_date	timestamp without time zone	Start time of a scheduled job.
repeat_interval	text	Period of a scheduled job.
event_queue_owner	character varying(128)	Not supported. Its value is **NULL**.
event_queue_name	character varying(128)	Not supported. Its value is **NULL**.

Name	Type	Description
event_queue_agent	character varying(523)	Not supported. Its value is **NULL**.
event_condition	character varying(4000)	Not supported. Its value is **NULL**.
event_rule	character varying(261)	Not supported. Its value is **NULL**.
file_watcher_owner	character varying(261)	Not supported. Its value is **NULL**.
file_watcher_name	character varying(261)	Not supported. Its value is **NULL**.
end_date	timestamp without time zone	End time of a scheduled job.
job_class	text	Name of the scheduled job class to which a scheduled job belongs.
enabled	boolean	Status of a scheduled job.
auto_drop	text	Status of the automatic deletion function of a scheduled job.
restart_on_recovery	character varying(5)	Not supported. Its value is **NULL**.
restart_on_failure	character varying(5)	Not supported. Its value is **NULL**.
state	"char"	Status of a scheduled job.
job_priority	numeric	Not supported. Its value is **NULL**.
run_count	numeric	Not supported. Its value is **NULL**.
uptime_run_count	numeric	Not supported. Its value is **NULL**.
max_runs	numeric	Not supported. Its value is **NULL**.
failure_count	smallint	Number of scheduled job failures.
uptime_failure_count	numeric	Not supported. Its value is **NULL**.

Name	Type	Description
max_failures	numeric	Maximum number of failures allowed before the status of a scheduled job is marked as broken.
retry_count	numeric	Not supported. Its value is **NULL**.
last_start_date	timestamp without time zone	Last time when a scheduled job was started.
last_run_duration	interval day to second(6)	Last execution duration of a scheduled job.
next_run_date	timestamp without time zone	Next execution time of a scheduled job.
schedule_limit	interval day to second(0)	Not supported. Its value is **NULL**.
max_run_duration	interval day to second(0)	Not supported. Its value is **NULL**.
logging_level	character varying(11)	Not supported. Its value is **NULL**.
store_output	character varying(5)	Specifies whether to store the output information of all scheduled jobs.
stop_on_window_close	character varying(5)	Not supported. Its value is **NULL**.
instance_stickiness	character varying(5)	Not supported. Its value is **NULL**.
raise_events	character varying(4000)	Not supported. Its value is **NULL**.
system	character varying(5)	Not supported. Its value is **NULL**.
job_weight	numeric	Not supported. Its value is **NULL**.
nls_env	character varying(4000)	Not supported. Its value is **NULL**.
source	character varying(128)	Not supported. Its value is **NULL**.
number_of_destinations	numeric	Not supported. Its value is **NULL**.
destination_owner	character varying(261)	Not supported. Its value is **NULL**.

Name	Type	Description
destination	text	Target name of a scheduled job.
credential_owner	character varying(128)	Not supported. Its value is **NULL**.
credential_name	text	Certificate name of a scheduled job
instance_id	oid	OID of the current database.
deferred_drop	character varying(5)	Not supported. Its value is **NULL**.
allow_runs_in_restricted_mode	character varying(5)	Not supported. Its value is **NULL**.
comments	text	Comments of a scheduled job.
flags	numeric	Not supported. Its value is **NULL**.
restartable	character varying(5)	Not supported. Its value is **NULL**.
has_constraints	character varying(5)	Not supported. Its value is **NULL**.
connect_credential_owner	character varying(128)	Not supported. Its value is **NULL**.
connect_credential_name	character varying(128)	Not supported. Its value is **NULL**.
fail_on_script_error	character varying(5)	Not supported. Its value is **NULL**.

12.3.12.21 ADM_SCHEDULER_PROGRAM_ARGS

ADM_SCHEDULER_PROGRAM_ARG displays parameters related to all programs in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-246 ADM_SCHEDULER_PROGRAM_ARGS columns

Name	Type	Description
owner	character varying(128)	Owner of the program to which the parameter belongs.

Name	Type	Description
program_name	character varying(128)	Name of the program to which the parameter belongs.
argument_name	character varying(128)	Parameter name.
argument_position	numeric	Position of the parameter in the parameter list.
argument_type	character varying(257)	Data type of a parameter.
metadata_attribute	character varying(19)	Not supported. Set it to **NULL**.
default_value	character varying(4000)	Default parameter value.
default_anydata_value	character varying(4000)	Not supported. Set it to **NULL**.
out_argument	character varying(5)	Reserved column. Set it to **NULL**.

12.3.12.22 ADM_SCHEDULER_PROGRAMS

ADM_SCHEDULER_PROGRAMS displays information about all programs that can be scheduled in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG_CATALOG and SYS schema.

Table 12-247 ADM_SCHEDULER_PROGRAMS columns

Name	Type	Description
owner	name	Owner of a scheduler program
program_name	text	Name of a scheduler program

Name	Type	Description
program_type	character varying(16)	Type of the scheduler. The options are as follows: • **PLSQL_BLOCK**: anonymous block of a stored procedure. • **STORED_PROCEDURE**: stored procedure that is saved. • **EXTERNAL_SCRIPT**: external script.
program_action	text	Action performed by a scheduler program
number_of_arguments	numeric	Number of scheduler program parameters
enabled	character varying(5)	Specifies whether a scheduler program is enabled.
comments	text	Comments of a scheduler program
detached	character varying(5)	Not supported. Its value is **NULL**.
schedule_limit	interval day to second(0)	Not supported. Its value is **NULL**.
priority	numeric	Not supported. Its value is **NULL**.
weight	numeric	Not supported. Its value is **NULL**.
max_runs	numeric	Not supported. Its value is **NULL**.
max_failures	numeric	Not supported. Its value is **NULL**.
max_run_duration	interval day to second(0)	Not supported. Its value is **NULL**.
has_constraints	character varying(5)	Not supported. Its value is **NULL**.
nls_env	character varying(4000)	Not supported. Its value is **NULL**.

12.3.12.23 ADM_SCHEDULER_RUNNING_JOBS

ADM_SCHEDULER_RUNNING_JOBS displays information about all **DBE_SCHEDULER** jobs that are being executed in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-248 ADM_SCHEDULER_RUNNING_JOBS columns

Name	Type	Description
owner	character varying(128)	Owner of a scheduler job
job_name	character varying(128)	Name of a scheduler job
job_subname	character varying(128)	Not supported. Set it to **NULL**.
job_style	character varying(17)	Action style of a scheduler job
detached	character varying(5)	Not supported. Set it to **NULL**.
session_id	numeric	ID of the session that executes a scheduler job
slave_process_id	numeric	Not supported. Set it to **NULL**.
slave_os_process_id	character varying(12)	ID of the process that executes a scheduler job
running_instance	numeric	Not supported. Set it to **NULL**.
resource_consumer_group	character varying(32)	Not supported. Set it to **NULL**.
elapsed_time	interval day to second(2)	Execution duration of a scheduler job
cpu_used	interval day to second(2)	Not supported. Set it to **NULL**.
destination_owner	character varying(261)	Not supported. Set it to **NULL**.
destination	character varying(261)	Target name of a scheduler job
credential_owner	character varying(128)	Not supported. Set it to **NULL**.
credential_name	character varying(128)	Certificate name of a scheduler job.

Name	Type	Description
log_id	numeric	Not supported. Set it to **NULL**.

12.3.12.24 ADM_SEGMENTS

ADM_SEGMENTS displays the storage space allocated to all segments in the database. It exists in the PG_CATALOG and SYS schemas. Only the system administrator can access this view. The Astore engine supports the segment-page storage, but information cannot be obtained from the system catalog.

Table 12-249 ADM_SEGMENTS columns

Name	Type	Description
owner	character varying(128)	Not supported. The value is **NULL**.
segment_name	character varying(128)	Not supported. The value is **NULL**.
partition_name	character varying(128)	Not supported. The value is **NULL**.
segment_type	character varying(18)	Not supported. The value is **NULL**.
segment_subtype	character varying(10)	Not supported. The value is **NULL**.
tablespace_name	character varying(30)	Not supported. The value is **NULL**.
header_file	numeric	Not supported. The value is **NULL**.
header_block	numeric	Not supported. The value is **NULL**.
bytes	numeric	Not supported. The value is **NULL**.
blocks	numeric	Not supported. The value is **NULL**.
extents	numeric	Not supported. The value is **NULL**.
initial_extent	numeric	Not supported. The value is **NULL**.
next_extent	numeric	Not supported. The value is **NULL**.

Name	Type	Description
min_extents	numeric	Not supported. The value is **NULL**.
max_extents	numeric	Not supported. The value is **NULL**.
max_size	numeric	Not supported. The value is **NULL**.
retention	character varying(7)	Not supported. The value is **NULL**.
minretention	numeric	Not supported. The value is **NULL**.
pct_increase	numeric	Not supported. The value is **NULL**.
freelists	numeric	Not supported. The value is **NULL**.
freelist_groups	numeric	Not supported. The value is **NULL**.
relative_fno	numeric	Not supported. The value is **NULL**.
buffer_pool	character varying(7)	Not supported. The value is **NULL**.
flash_cache	character varying(7)	Not supported. The value is **NULL**.
cell_flash_cache	character varying(7)	Not supported. The value is **NULL**.
inmemory	character varying(8)	Not supported. The value is **NULL**.
inmemory_priority	character varying(8)	Not supported. The value is **NULL**.
inmemory_distribute	character varying(15)	Not supported. The value is **NULL**.
inmemory_duplicate	character varying(13)	Not supported. The value is **NULL**.
inmemory_compression	character varying(17)	Not supported. The value is **NULL**.
cellmemory	character varying(24)	Not supported. The value is **NULL**.

12.3.12.25 ADM_SEQUENCES

ADM_SEQUENCES displays information about all sequences in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG_CATALOG and SYS schema.

Table 12-250 ADM_SEQUENCES columns

Name	Type	Description
sequence_owner	character varying(64)	Owner of a sequence.
sequence_name	character varying(64)	Name of the sequence.
min_value	int16	Minimum value of the sequence.
max_value	int16	Maximum value of the sequence.
increment_by	int16	Increment of the sequence.
last_number	int16	Value of the previous sequence.
cache_size	int16	Size of the sequence disk cache.
cycle_flag	character(1)	Specifies whether a sequence is a cyclic sequence. Value range: • **Y**: It is a cyclic sequence. • **N**: It is not a cyclic sequence.

12.3.12.26 ADM_SOURCE

ADM_SOURCE displays the definition information about all stored procedures, functions, triggers, and packages in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG_CATALOG and SYS schema.

Table 12-251 ADM_SOURCE columns

Name	Type	Description
owner	name	Object owner.
name	name	Object name.

Name	Type	Description
type	name	Object type. Its values can be **function**, **package**, **package body**, **procedure**, and **trigger**.
line	numeric	Number of the line in the definition information.
text	text	Text source of the storage object.
origin_con_id	character varying(256)	Not supported. Its value is **0**.

12.3.12.27 ADM_SYNONYMS

ADM_SYNONYMS displays all synonyms in the database. This view is accessible only to system administrators. This view exists in both PG_CATALOG and SYS schema.

Table 12-252 ADM_SYNONYMS columns

Name	Type	Description
owner	text	Owner of a synonym. The owner of the PUBLIC synonym is PUBLIC.
schema_name	text	Name of the schema to which the synonym belongs. The name of the schema to which the PUBLIC synonym belongs is NULL.
synonym_name	text	Synonym name.
table_owner	text	Owner of the associated object. Although the column is called **table_owner**, its associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.
table_name	text	Name of the associated object. Although the column is called **table_name**, its associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.
table_schema_name	text	Schema name of the associated object. Although the column is called **table_schema_name**, its associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.

12.3.12.28 ADM_TAB_COL_STATISTICS

ADM_TAB_COL_STATISTICS displays column statistics and histogram information extracted from ADM_TAB_COLUMNS. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG_CATALOG and SYS schema. The values of the **LOW_VALUE** and **HIGH_VALUE** columns in this view are different from those in database A due to different underlying table structures. When the value of **LOW_VALUE** is a high-frequency value, the value of **LOW_VALUE** in GaussDB is the second smallest value. When **HIGH_VALUE** is a high-frequency value, **HIGH_VALUE** of GaussDB is the second highest value. The value of the **HISTOGRAM** column is different from that of database A due to different statistical methods. GaussDB supports only two types of histograms: frequency and equi-width. The value of the **SCOPE** column in GaussDB is different from that in database A because GaussDB does not support global temporary table statistics. GaussDB database supports only local temporary table statistics, and the default value is **SHARED**.

Table 12-253 ADM_TAB_COL_STATISTICS columns

Name	Type	Description
owner	character varying(128)	Table owner.
table_name	character varying(128)	Table name.
column_name	character varying(128)	Column name.
num_distinct	numeric	Number of different values in a column.
low_value	raw	Low value in a column.
high_value	raw	High value in a column.
density	numeric	• If there is a histogram on **COLUMN_NAME**, this column displays the selectivity of values in the histogram that span less than two endpoints. It does not represent the selectivity of values that span two or more endpoints. • If no histogram is available on **COLUMN_NAME**, the value of this column is **1/NUM_DISTINCT**.
num_nulls	numeric	Number of empty values in a column.

Name	Type	Description
num_buckets	numeric	Number of buckets in the histogram of a column.
sample_size	numeric	Sample size used to analyze a column.
last_analyzed	timestamp(0) without time zone	Date when a column was last analyzed. Database restart is not supported. Otherwise, data loss will occur.
global_stats	character varying(3)	Not supported. Its value is **NULL**.
user_stats	character varying(3)	Not supported. Its value is **NULL**.
notes	character varying(99)	Not supported. Its value is **NULL**.
avg_col_len	numeric	Average length of a column, in bytes.
histogram	character varying(15)	Specifies whether the histogram exists and the type of the histogram. ● **NONE**: no histogram. ● **FREQUENCY**: frequency histogram. ● **EQUI-WIDTH**: equal-width histogram.
scope	character varying(7)	For statistics collected on any table other than global temporary tables, the value is **SHARED** (indicating that the statistics are shared among all sessions).
schema	character varying(64)	Name of the namespace to which the column belongs.

12.3.12.29 ADM_TAB_COLS

ADM_TAB_COLS displays information about table and view columns. Each column of each table and view in the database has a row of data in ADM_TAB_COLS. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG_CATALOG

and SYS schema. The number of rows displayed in this view is the same as that in the ADM_TAB_COLUMNS view. Only columns are different.

Table 12-254 ADM_TAB_COLS columns

Name	Type	Description
owner	character varying(64)	Owner of a table or view.
table_name	character varying(128)	Name of the table or view.
column_name	character varying(128)	Column name.
data_type	character varying(128)	Data type of a column, which can be a user-defined data type.
data_type_mod	character varying(3)	Not supported. Its value is **NULL**.
data_type_owner	character varying(128)	Owner of the data type of a column.
data_length	numeric	Length of a column, in bytes.
data_precision	numeric	Precision of a data type. This column is valid for the numeric data type and **NULL** for other data types.
data_scale	numeric	Number of decimal places. This column is valid for the numeric data type and is set to **0** for other data types.
nullable	character varying(1)	Specifies whether a column can be empty (**n** for the primary key constraint and non-null constraint).
column_id	numeric	Sequence number of a column when a table is created.
default_length	numeric	Length of the default value of a column, in bytes.
data_default	text	Default value of a column.
num_distinct	numeric	Number of different values in a column.
low_value	raw	Minimum value in a column.
high_value	raw	Maximum value in a column.
density	numeric	Column density.

Name	Type	Description
num_nulls	numeric	Number of empty values in a column.
num_buckets	numeric	Number of buckets in the histogram of a column.
last_analyzed	timestamp(0) without time zone	Last analysis date.
sample_size	numeric	Sample size used to analyze a column.
character_set_name	character varying(44)	Not supported. Its value is **NULL**.
char_col_decl_length	numeric	Declaration length of a column of the character type.
global_stats	character varying(3)	Not supported. Its value is **NO**.
user_stats	character varying(3)	Not supported. Its value is **NO**.
avg_col_len	numeric	Average length of a column, in bytes.
char_length	numeric	Column length (in bytes) which is valid only for the varchar, nvarchar2, bpchar, and char types.
char_used	character varying(1)	Not supported. Set it to **B** if the data type is varchar, nvarchar2, bpchar, or char, and to **NULL** for other data types.
v80_fmt_image	character varying(3)	Not supported. Its value is **NULL**.
data_upgraded	character varying(3)	Not supported. Its value is **YES**.
hidden_column	character varying(3)	Not supported. Its value is **NULL**.
virtual_column	character varying	Specifies whether a column is a virtual column (a generated column). • **YES** • **NO**

Name	Type	Description
segment_column_id	numeric	Not supported. Its value is **NULL**.
internal_column_id	numeric	Internal sequence number of a column. The value is the same as that of **COLUMN_ID**.
histogram	character varying(15)	Specifies whether the histogram exists and the type of the histogram. • **NONE**: no histogram. • **FREQUENCY**: frequency histogram. • **EQUI_WIDTH**: equal-width histogram.
qualified_col_name	character varying(64)	Qualified column name, which is the same as **COLUMN_NAME**.
user_generated	character varying(3)	Not supported. Its value is **YES**.
default_on_null	character varying(3)	Not supported. Its value is **NULL**.
identity_column	character varying(3)	Not supported. Its value is **NULL**.
sensitive_column	character varying(3)	Not supported. Its value is **NULL**.
evaluation_edition	character varying(128)	Not supported. Its value is **NULL**.
unusable_before	character varying(128)	Not supported. Its value is **NULL**.
unusable_beginning	character varying(128)	Not supported. Its value is **NULL**.
collation	character varying(100)	Collation rule of a column. This column conflicts with reserved keywords. Therefore, add the view name when calling this column.
collated_column_id	numeric	Not supported. Its value is **NULL**.
schema	character varying(64)	Name of the namespace to which the column belongs.

12.3.12.30 ADM_TAB_COLUMNS

ADM_TAB_COLUMNS displays information about the columns of tables and views. Each column of each table and view in the database has a row of data in ADM_TAB_COLUMNS. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG_CATALOG and SYS schema.

Table 12-255 ADM_TAB_COLUMNS columns

Name	Type	Description
owner	character varying(64)	Table owner.
table_name	character varying(64)	Table name.
column_name	character varying(64)	Column name.
data_type	character varying(128)	Data type of a column.
data_type_mod	character varying(3)	Not supported. Its value is **NULL**.
data_type_owner	character varying(128)	Owner of the data type of a column.
data_length	integer	Length of a column, in bytes.
data_precision	integer	Precision of a data type. This column is valid for the numeric data type and **NULL** for other data types.
data_scale	integer	Number of decimal places. This column is valid for the numeric data type and is set to **0** for other data types.
nullable	bpchar	Specifies whether the column can be null. Value range: • **y**: yes. • **n**: no. For primary key constraints and non-null constraints, the value is **n**.
column_id	integer	Sequence number of a column when a table is created.
default_length	numeric	Length of the default value of a column, in bytes. This column is left blank if no default value left.
data_default	text	Default value of a column.
num_distinct	numeric	Number of different values in a column.

Name	Type	Description
low_value	raw	Minimum value in a column.
high_value	raw	Maximum value in a column.
density	numeric	Column density.
num_nulls	numeric	Number of empty values in a column.
num_buckets	numeric	Number of buckets in the histogram of a column.
last_analyzed	timestamp(0) without time zone	Last analysis date.
sample_size	numeric	Sample size used to analyze a column.
character_set_name	character varying(44)	Not supported. Its value is **NULL**.
char_col_decl_length	numeric	Declaration length of a column of the character type.
global_stats	character varying(3)	Not supported. Its value is **NO**.
user_stats	character varying(3)	Not supported. Its value is **NO**.
avg_col_len	numeric	Average length of a column, in bytes.
char_length	numeric	Column length (in bytes) which is valid only for the varchar, nvarchar2, bpchar, and char types.
char_used	character varying(1)	Not supported. Set it to **B** if the data type is varchar, nvarchar2, bpchar, or char. For other data types, set it to **NULL**.
v80_fmt_image	character varying(3)	Not supported. Its value is **NULL**.
data_upgraded	character varying(3)	Not supported. Its value is **YES**.
histogram	character varying(15)	Specifies whether the histogram exists and the type of the histogram. • **NONE**: no histogram. • **FREQUENCY**: frequency histogram. • **EQUI_WIDTH**: equal-width histogram.
default_on_null	character varying(3)	Not supported. Its value is **NULL**.
identity_column	character varying(3)	Not supported. Its value is **NULL**.

Name	Type	Description
sensitive_column	character varying(3)	Not supported. Its value is **NULL**.
evaluation_edition	character varying(128)	Not supported. Its value is **NULL**.
unusable_before	character varying(128)	Not supported. Its value is **NULL**.
unusable_beginning	character varying(128)	Not supported. Its value is **NULL**.
collation	character varying(100)	Collation rule of a column. This column conflicts with reserved keywords. Therefore, add the view name when calling this column.
comments	text	Comment of a column.
schema	character varying(64)	Name of the namespace to which the column belongs.

12.3.12.31 ADM_TAB_COMMENTS

ADM_TAB_COMMENTS displays comments about all tables and views in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-256 ADM_TAB_COMMENTS columns

Name	Type	Description
owner	character varying(64)	Owner of a table or view.
table_name	character varying(64)	Name of a table or view.
comments	text	Comments.
schema	character varying(64)	Name of the namespace to which the table belongs.

12.3.12.32 ADM_TAB_HISTOGRAMS

ADM_TAB_HISTOGRAMS displays the histogram information about all tables and views in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-257 ADM_TAB_HISTOGRAMS columns

Name	Type	Description
owner	character varying(128)	Table owner.
table_name	character varying(128)	Table name.
column_name	character varying(4000)	Column name or attribute of an object column.
endpoint_number	numeric	Bucket ID of the histogram.
endpoint_value	numeric	Not supported. Set it to **NULL**.
endpoint_actual_value	character varying(4000)	Actual value of the bucket endpoint.
endpoint_actual_value_raw	raw	Not supported. Set it to **NULL**.
endpoint_repeat_count	numeric	Not supported. Set it to **NULL**.
scope	character varying(7)	Not supported. Set it to **SHARED**.

12.3.12.33 ADM_TAB_STATISTICS

ADM_TAB_STATISTICS displays optimizer statistics for all tables in the database. This view exists in the PG_CATALOG and SYS schemas. By default, only the system administrator can access this view. Common users can access the view only after being authorized.

Table 12-258 ADM_TAB_STATISTICS columns

Name	Type	Description
owner	character varying(128)	Object owner.
table_name	character varying(128)	Table name.
partition_name	character varying(128)	Not supported. Set it to **NULL**.
partition_position	numeric	Not supported. Set it to **NULL**.
subpartition_name	character varying(128)	Not supported. Set it to **NULL**.

Name	Type	Description
subpartition_position	numeric	Not supported. Set it to **NULL**.
object_type	character varying(12)	Object type. • **TABLE** • **PARTITION** • **SUBPARTITION**
num_rows	numeric	Number of rows in an object.
blocks	numeric	Not supported. Set it to **NULL**.
empty_blocks	numeric	Not supported. Set it to **NULL**.
avg_space	numeric	Not supported. Set it to **NULL**.
chain_cnt	numeric	Not supported. Set it to **NULL**.
avg_row_len	integer	Average row length, including the row overhead.
avg_space_freelist_blocks	numeric	Not supported. Set it to **NULL**.
num_freelist_blocks	numeric	Not supported. Set it to **NULL**.
avg_cached_blocks	numeric	Not supported. Set it to **NULL**.
avg_cache_hit_ratio	numeric	Not supported. Set it to **NULL**.
im_imcu_count	numeric	Not supported. Set it to **NULL**.
im_block_count	numeric	Not supported. Set it to **NULL**.
im_stat_update_time	timestamp(6) without time zone	Not supported. Set it to **NULL**.
scan_rate	numeric	Not supported. Set it to **NULL**.
sample_size	numeric	Number of samples used for analyzing a table.
last_analyzed	timestamp with time zone	Date when a table was last analyzed. Database restart is not supported. Otherwise, data loss will occur.
global_stats	character varying(3)	Not supported. Set it to **NULL**.
user_stats	character varying(3)	Not supported. Set it to **NULL**.

Name	Type	Description
stattype_locked	character varying(5)	Not supported. Set it to **NULL**.
stale_stats	character varying(7)	Not supported. Set it to **NULL**.
notes	character varying(25)	Not supported. Set it to **NULL**.
scope	character varying(7)	Not supported. Set it to **SHARED**.

12.3.12.34 ADM_TAB_STATS_HISTORY

ADM_TAB_STATS_HISTORY provides historical statistics about all tables in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-259 ADM_TAB_STATS_HISTORY columns

Name	Type	Description
owner	character varying(128)	Object owner.
table_name	character varying(128)	Table name.
partition_name	character varying(128)	Not supported. Set it to **NULL**.
subpartition_name	character varying(128)	Not supported. Set it to **NULL**.
stats_update_time	timestamp(6) with time zone	Time when statistics are updated. Database restart is not supported. Otherwise, data loss will occur.

12.3.12.35 ADM_TABLES

ADM_TABLES displays all tables in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-260 ADM_TABLES columns

Name	Type	Description
owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
tablespace_name	character varying(64)	Tablespace name of the table
dropped	character varying	Specifies whether the current table is deleted. ● **YES**: The record is deleted. ● **NO**: The record is not deleted.
num_rows	numeric	Estimated number of rows in the table
status	character varying(8)	Specifies whether the current table is valid. ● **VALID**: The current table is valid. ● **UNUSABLE**: The current table is unavailable.
sample_size	numeric	Number of samples used for analyzing the table
temporary	character(1)	Specifies whether a table is a temporary table. ● **Y**: The table is a temporary table. ● **N**: The table is not a temporary table.
pct_free	numeric	Minimum percentage of free space in a block
ini_trans	numeric	Initial number of transactions
max_trans	numeric	Maximum number of transactions
avg_row_len	integer	Average number of bytes in each row

Name	Type	Description
partitioned	character varying(3)	Specifies whether a table is a partitioned table. ● **YES**: The table is a partitioned table. ● **NO**: The table is not a partitioned table.
last_analyzed	timestamp with time zone	Last time when the table was analyzed. Database restart is not supported. Otherwise, data loss will occur.
row_movement	character varying(8)	Specifies whether to allow partition row movement. ● **ENABLED**: The partition row movement is allowed. ● **DISABLED**: The partition row movement is not allowed.
compression	character varying(8)	Specifies whether to enable a table compression. ● **ENABLED**: A table compression is enabled. ● **DISABLED**: A table compression is disabled.
duration	character varying(15)	Time elapsed when a temporary table is processed ● **NULL**: The table is not a temporary table. ● **sys$session**: The table is a temporary session table. ● **sys$transaction**: The table is a temporary transaction table.

Name	Type	Description
logical_replication	character varying(8)	Specifies whether logical replication is enabled for a table. • **ENABLED**: Logical replication is enabled. • **DISABLED**: Logical replication is disabled.
external	character varying(3)	Specifies whether the table is an external table. • **YES**: The table is an external table. • **NO**: The table is not an external table.
logging	character varying(3)	Specifies whether changes to a table are logged. • **YES**: Logs are recorded for table changes. • **NO**: Logs are not recorded for table changes.
default_collation	character varying(100)	Default collation of a table
degree	character varying(10)	Number of instances in a scanned table
table_lock	character varying(8)	Specifies whether to enable a table lock. • **ENABLED**: The table lock is enabled. • **DISABLED**: The table lock is disabled.
nested	character varying(3)	Specifies whether a table is a nested table. • **YES**: The table is a nested table. • **NO**: The table is not a nested table.
buffer_pool	character varying(7)	Default buffer pool of a table
flash_cache	character varying(7)	Not supported. The value is **NULL**.

Name	Type	Description
cell_flash_cache	character varying(7)	Not supported. The value is **NULL**.
skip_corrupt	character varying(8)	Specifies whether to skip corrupted blocks during table scanning. ● **ENABLED**: The corrupted block is skipped. ● **DISABLED**: The corrupted block is not skipped.
has_identity	character varying(3)	Specifies whether a table has an identifier column. ● **YES**: There is an identifier column. ● **NO**: There is no identifier column.
segment_created	character varying(3)	Specifies whether a table segment has been created. ● **YES**: The table segment has been created. ● **NO**: The table segment is not created.
monitoring	character varying(3)	Specifies whether to monitor the modification of a table. ● **YES**: The modification of the table is monitored. ● **NO**: The modification of the table is not monitored.
cluster_name	character varying(128)	Not supported. The value is **NULL**.
iot_name	character varying(128)	Not supported. The value is **NULL**.
pct_used	numeric	Not supported. The value is **NULL**.

Name	Type	Description
initial_extent	numeric	Not supported. The value is **NULL**.
next_extent	numeric	Not supported. The value is **NULL**.
min_extents	numeric	Not supported. The value is **NULL**.
max_extents	numeric	Not supported. The value is **NULL**.
pct_increase	numeric	Not supported. The value is **NULL**.
freelists	numeric	Not supported. The value is **NULL**.
freelist_groups	numeric	Not supported. The value is **NULL**.
backed_up	character varying(1)	Not supported. The value is **NULL**.
blocks	numeric	Not supported. The value is **NULL**.
empty_blocks	numeric	Not supported. The value is **NULL**.
avg_space	numeric	Not supported. The value is **NULL**.
chain_cnt	numeric	Not supported. The value is **NULL**.
avg_space_freelist_blocks	numeric	Not supported. The value is **NULL**.
num_freelist_blocks	numeric	Not supported. The value is **NULL**.
instances	character varying(10)	Not supported. The value is **NULL**.
cache	character varying(5)	Not supported. The value is **NULL**.
iot_type	character varying(12)	Not supported. The value is **NULL**.
secondary	character varying(1)	Not supported. The value is **NULL**.
global_stats	character varying(3)	Not supported. The value is **NULL**.

Name	Type	Description
user_stats	character varying(3)	Not supported. The value is **NULL**.
cluster_owner	character varying(30)	Not supported. The value is **NULL**.
dependencies	character varying(8)	Not supported. The value is **NULL**.
compression_for	character varying(30)	Not supported. The value is **NULL**.
read_only	character varying(3)	Not supported. The value is **NULL**.
result_cache	character varying(7)	Not supported. The value is **NULL**.
clustering	character varying(3)	Not supported. The value is **NULL**.
activity_tracking	character varying(23)	Not supported. The value is **NULL**.
dml_timestamp	character varying(25)	Not supported. The value is **NULL**.
container_data	character varying(3)	Not supported. The value is **NULL**.
inmemory_priority	character varying(8)	Not supported. The value is **NULL**.
inmemory_distribute	character varying(15)	Not supported. The value is **NULL**.
inmemory_compression	character varying(17)	Not supported. The value is **NULL**.
inmemory_duplicate	character varying(13)	Not supported. The value is **NULL**.
duplicated	character varying(1)	Not supported. The value is **NULL**.
sharded	character varying(1)	Not supported. The value is **NULL**.
hybrid	character varying(3)	Not supported. The value is **NULL**.
cellmemory	character varying(24)	Not supported. The value is **NULL**.
containers_default	character varying(3)	Not supported. The value is **NULL**.

Name	Type	Description
container_map	character varying(3)	Not supported. The value is **NULL**.
extended_data_link	character varying(3)	Not supported. The value is **NULL**.
extended_data_link_map	character varying(3)	Not supported. The value is **NULL**.
inmemory_service	character varying(12)	Not supported. The value is **NULL**.
inmemory_service_name	character varying(1000)	Not supported. The value is **NULL**.
container_map_object	character varying(3)	Not supported. The value is **NULL**.
memoptimize_read	character varying(8)	Not supported. The value is **NULL**.
memoptimize_write	character varying(8)	Not supported. The value is **NULL**.
has_sensitive_column	character varying(3)	Not supported. The value is **NULL**.
admit_null	character varying(3)	Not supported. The value is **NULL**.
data_link_dml_enabled	character varying(3)	Not supported. The value is **NULL**.
object_id_type	character varying(16)	Not supported. The value is **NULL**.
table_type_owner	character varying(128)	Not supported. The value is **NULL**.
table_type	character varying(128)	Not supported. The value is **NULL**.
compress_for	character varying(30)	Not supported. The value is **NULL**.

12.3.12.36 ADM_TABLESPACES

ADM_TABLESPACES displays information about all tablespaces. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas. The logical structure features of GaussDB are different from those of the A database.

Table 12-261 ADM_TABLESPACES columns

Name	Type	Description
tablespace_name	character varying(64)	Tablespace name.
block_size	numeric	Not supported. The value is **NULL**.
initial_extent	numeric	Not supported. The value is **NULL**.
next_extent	numeric	Not supported. The value is **NULL**.
min_extents	numeric	Not supported. The value is **NULL**.
max_extents	numeric	Not supported. The value is **NULL**.
max_size	numeric	Not supported. The value is **NULL**.
pct_increase	numeric	Not supported. The value is **NULL**.
min_extlen	numeric	Not supported. The value is **NULL**.
contents	character varying(9)	Not supported. The value is **NULL**.
status	character varying(9)	Not supported. The value is **ONLINE**.
logging	character varying(9)	Not supported. The value is **NULL**.
force_logging	character varying(3)	Not supported. The value is **NULL**.
extent_management	character varying(10)	Not supported. The value is **NULL**.
allocation_type	character varying(9)	Not supported. The value is **NULL**.
plugged_in	character varying(3)	Not supported. The value is **NULL**.
segment_space_management	character varying(6)	Not supported. The value is **NULL**.
def_tab_compression	character varying(8)	Not supported. The value is **NULL**.
retention	character varying(11)	Not supported. The value is **NULL**.

Name	Type	Description
bigfile	character varying(3)	Not supported. The value is **NULL**.
predicate_evaluation	character varying(7)	Not supported. The value is **NULL**.
encrypted	character varying(3)	Not supported. The value is **NULL**.
compress_for	character varying(30)	Not supported. The value is **NULL**.
def_inmemory	character varying(8)	Not supported. The value is **NULL**.
def_inmemory_priority	character varying(8)	Not supported. The value is **NULL**.
def_inmemory_distribute	character varying(15)	Not supported. The value is **NULL**.
def_inmemory_compression	character varying(17)	Not supported. The value is **NULL**.
def_inmemory_duplicate	character varying(13)	Not supported. The value is **NULL**.
shared	character varying(12)	Not supported. The value is **NULL**.
def_index_compression	character varying(8)	Not supported. The value is **NULL**.
index_compress_for	character varying(13)	Not supported. The value is **NULL**.
def_cellmemory	character varying(14)	Not supported. The value is **NULL**.
def_inmemory_service	character varying(12)	Not supported. The value is **NULL**.
def_inmemory_service_name	character varying(1000)	Not supported. The value is **NULL**.
lost_write_protect	character varying(7)	Not supported. The value is **NULL**.
chunk_tablespace	character varying(1)	Not supported. The value is **NULL**.

12.3.12.37 ADM_TRIGGERS

ADM_TRIGGERS displays information about triggers in the database. By default, only the system administrator can access this view. Common users can access the

view only after being authorized. This view exists in both PG_CATALOG and SYS schema.

Table 12-262 ADM_TRIGGERS columns

Name	Type	Description
owner	character varying(128)	Trigger owner.
trigger_name	character varying(64)	Trigger name.
trigger_type	character varying	Time when a trigger is triggered. Value range: **before statement**, **before each row**, **after statement**, **after each row**, and **instead of**.
triggering_event	character varying	Event that triggers a trigger. Value range: **update**, **insert**, **delete**, and **truncate**.
table_owner	character varying(64)	Owner of the table that defines a trigger.
base_object_type	character varying(18)	Defines the basic object of a trigger. Value range: **table** and **view**.
table_name	character varying(64)	Name of the table or view that defines a trigger.
column_name	character varying(4000)	Not supported. Its value is **NULL**.
referencing_name	character varying(422)	Not supported. Its value is **referencing new as new old as old**.
when_clause	character varying(4000)	Content of the WHEN clause. **trigger_body** can be executed only when the value is **true**.
status	character varying(64)	Trigger status: • **O**: The trigger is enabled in origin or local mode. • **D**: The trigger is disabled. • **R**: The trigger is enabled in replica mode. • **A**: The trigger is always enabled.
description	character varying(4000)	Trigger description, which is used to rebuild a trigger creation statement.
action_type	character varying(11)	Action type of a trigger, which only supports **call**.
trigger_body	text	Statement executed when a trigger is triggered.

Name	Type	Description
crossedition	character varying(7)	Not supported. Its value is **NULL**.
before_statement	character varying(3)	Not supported. Its value is **NULL**.
before_row	character varying(3)	Not supported. Its value is **NULL**.
after_row	character varying(3)	Not supported. Its value is **NULL**.
after_statement	character varying(3)	Not supported. Its value is **NULL**.
instead_of_row	character varying(3)	Not supported. Its value is **NULL**.
fire_once	character varying(3)	Not supported. Its value is **NULL**.
apply_server_only	character varying(3)	Not supported. Its value is **NULL**.

12.3.12.38 ADM_TYPE_ATTRS

ADM_TYPE_ATTRS displays the attributes of the current database object type. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in both PG_CATALOG and SYS schema.

Table 12-263 ADM_TYPE_ATTRS columns

Name	Type	Description
owner	oid	Owner of the type.
type_name	name	Data type name.
attr_name	name	Column name.
attr_type_mod	integer	Type-specific data supplied at the table creation time (for example, the maximum length of a **varchar** column). This column is used as the third parameter when passing to type-specific input functions and length coercion functions. The value will generally be **–1** for types that do not need ATTTYPMOD.
attr_type_owner	oid	Owner of an attribute of this type.

Name	Type	Description
attr_type_name	name	Data type attribute name, which records the type name after conversion.
length	smallint	Number of bytes in the internal representation of the type for a fixed-size type. It is a negative number for a variable-length type. • The value **-1** indicates a "varlena" type (one that has a length word). • The value **-2** indicates a NULL-terminated C string.
precision	integer	Precision of the numeric type.
scale	integer	Range of the numeric type
character_set_name	character(1)	Character set name of an attribute (**c** or **n**). • **c**: CHAR_CS. • **n**: NCHAR_CS.
attr_no	smallint	Attribute number
inherited	character(1)	Specifies whether the attribute is inherited from the super type (**Y** or **N**).
attr_length	integer	Type-specific data supplied at the table creation time (for example, the maximum length of a **varchar** column). The raw type is not recorded due to kernel implementation.

12.3.12.39 ADM_TYPES

ADM_TYPES describes all object types in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-264 ADM_TYPES columns

Name	Type	Description
owner	character varying(128)	Owner of the type.
type_name	character varying(128)	Type name.
type_oid	raw	Type OID.
typecode	character varying(128)	Type code.
attributes	numeric	Number of attributes in a type.
methods	numeric	Not supported. Set it to **0**.

Name	Type	Description
predefined	character varying(3)	Specifies whether the type is a predefined type.
incomplete	character varying(3)	Specifies whether the type is an incomplete type.
final	character varying(3)	Not supported. Set it to **NULL**.
instantiable	character varying(3)	Not supported. Set it to **NULL**.
persistable	character varying(3)	Not supported. Set it to **NULL**.
supertype_owner	character varying(128)	Not supported. Set it to **NULL**.
supertype_name	character varying(128)	Not supported. Set it to **NULL**.
local_attributes	numeric	Not supported. Set it to **NULL**.
local_methods	numeric	Not supported. Set it to **NULL**.
typeid	raw	Not supported. Set it to **NULL**.

12.3.12.40 ADM_VIEWS

ADM_VIEWS displays views in the database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-265 ADM_VIEWS columns

Name	Type	Description
owner	character varying(64)	Owner of a view.
view_name	character varying(64)	Name of the view.
text	text	Text of a view.
text_length	integer	Text length of a view.

Name	Type	Description
text_vc	character varying(4000)	View creation statement. This column may truncate the view text. The BEQUEATH clause will not appear as part of the **TEXT_VC** column in this view.
type_text_length	numeric	Not supported. Set it to **NULL**.
type_text	character varying(4000)	Not supported. Set it to **NULL**.
oid_text_length	numeric	Not supported. Set it to **NULL**.
oid_text	character varying(4000)	Not supported. Set it to **NULL**.
view_type_owner	character varying(128)	Not supported. Set it to **NULL**.
view_type	character varying(128)	Not supported. Set it to **NULL**.
superview_name	character varying(128)	Not supported. Set it to **NULL**.
editioning_view	character varying(1)	Not supported. Set it to **NULL**.
read_only	character varying(1)	Not supported. Set it to **NULL**.
container_data	character varying(1)	Not supported. Set it to **NULL**.
bequeath	character varying(12)	Not supported. Set it to **NULL**.
origin_con_id	character varying(256)	Not supported. Set it to **NULL**.
default_collation	character varying(100)	Not supported. Set it to **NULL**.
containers_default	character varying(3)	Not supported. Set it to **NULL**.
container_map	character varying(3)	Not supported. Set it to **NULL**.
extended_data_link	character varying(3)	Not supported. Set it to **NULL**.

Name	Type	Description
extended_data_link_map	character varying(3)	Not supported. Set it to **NULL**.
has_sensitive_column	character varying(3)	Not supported. Set it to **NULL**.
admit_null	character varying(3)	Not supported. Set it to **NULL**.
pdb_local_only	character varying(3)	Not supported. Set it to **NULL**.

12.3.12.41 DB_ARGUMENTS

DB_ARGUMENTS displays parameter information about stored procedures and functions accessible to the current user. This view exists in both PG_CATALOG and SYS schema. This view is accessible to all users and displays all information accessible to the current user.

Table 12-266 DB_ARGUMENTS columns

Name	Type	Description
owner	character varying(128)	Owner of a function or stored procedure.
object_name	character varying(128)	Name of a function or stored procedure.
package_name	character varying(128)	Package name.
object_id	oid	OID of a function or stored procedure.
overload	character varying(40)	*n*th overloaded function of the name.
subprogram_id	numeric	Location of a function or stored procedure in a package.
argument_name	character varying(128)	Parameter name.
position	numeric	Position of the parameter in the parameter list. The value is **0** for the return value of the function by default.

Name	Type	Description
sequence	numeric	Sequence of a parameter, which starts from 1, with the return type before all parameters.
data_level	numeric	Nesting depth of parameters of the composite type. The value is fixed at **0** because only one line is displayed for each parameter.
data_type	character varying(30)	Data type of a parameter.
defaulted	character varying(1)	Specifies whether a parameter has a default value: • **Y**: yes. • **N**: no.
default_value	text	Not supported. Its value is **NULL**.
default_length	numeric	Not supported. Its value is **NULL**.
in_out	character varying(9)	Input and output attributes of a parameter: • **IN**: input parameter. • **OUT**: output parameter. • **IN_OUT**: input and output parameters. • **VARIADIC**: VARIADIC parameter.
data_length	numeric	Not supported. Its value is **NULL**.
data_precision	numeric	Not supported. Its value is **NULL**.
data_scale	numeric	Not supported. Its value is **NULL**.

Name	Type	Description
radix	numeric	Radix of a number, which is **10** when the data type is smallint, integer, bigint, numeric, or float. For other data types, set this column to **NULL**.
character_set_name	character varying(44)	Not supported. Its value is **NULL**.
type_owner	character varying(128)	Owner of the data type.
type_name	character varying(128)	Parameter type name. Only the customized type is displayed.
type_subname	character varying(128)	Not supported. Its value is **NULL**.
type_link	character varying(128)	Not supported. Its value is **NULL**.
type_object_type	character varying(7)	Type of **type_name**: • **TABLE**: The parameter is of the table type. • **VIEW**: The parameter is of the view type. • For other data types, the value is **NULL**.
pls_type	character varying(128)	Name of the PL/SQL type for parameters of the number type. Otherwise, this column is empty.
char_length	numeric	Not supported. Its value is **NULL**.
char_used	character varying(1)	Not supported. Set it to **B** if the data type is varchar, nvarchar2, bpchar, or char, and to **NULL** for other data types.
origin_con_id	character varying(256)	Not supported. Its value is **0**.

12.3.12.42 DB_ALL_TABLES

DB_ALL_TABLES displays tables or views accessible to the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-267 DB_ALL_TABLES columns

Name	Type	Description
owner	name	Owner of a table or view.
table_name	name	Name of a table or view.
tablespace_name	name	Tablespace where a table or view is located.

12.3.12.43 DB_COL_COMMENTS

DB_COL_COMMENTS displays comment information about table columns accessible to the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-268 DB_COL_COMMENTS columns

Name	Type	Description
owner	character varying(64)	Table owner.
table_name	character varying(64)	Table name.
column_name	character varying(64)	Column name.
comments	text	Comments.
schema	character varying(64)	Name of the namespace to which the column belongs.

12.3.12.44 DB_COLL_TYPES

DB_COLL_TYPES displays information about collection types that can be accessed by the current user. By default, they are accessible to all users. This view exists in the PG_CATALOG and SYS schemas.

Table 12-269 DB_COLL_TYPES columns

Name	Type	Description
owner	character varying(128)	Owner of a collection.
type_name	character varying(128)	Name of a collection.
coll_type	character varying(128)	Description of a collection.

Name	Type	Description
upper_bound	numeric	Not supported. Set it to **NULL**.
elem_type_mod	character varying(7)	Type modifier of an element.
elem_type_owner	character varying(128)	Owner of the element type on which the collection is based. This parameter is mainly used for user-defined types.
elem_type_name	character varying(128)	Name of the data type or user-defined type on which the collection is based.
length	numeric	Not supported. Set it to **NULL**.
precision	numeric	Not supported. Set it to **NULL**.
scale	numeric	Not supported. Set it to **NULL**.
character_set_name	character varying(44)	Not supported. Set it to **NULL**.
elem_storage	character varying(7)	Not supported. Set it to **NULL**.
nulls_stored	character varying(3)	Not supported. Set it to **NULL**.
char_used	character varying(1)	Not supported. Set it to **NULL**.

12.3.12.45 DB_CONS_COLUMNS

DB_CONS_COLUMNS displays information about constraint columns accessible to the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-270 DB_CONS_COLUMNS columns

Name	Type	Description
constraint_name	character varying(64)	Constraint name.
table_name	character varying(64)	Name of a constraint-related table.

Name	Type	Description
column_name	character varying(64)	Name of a constraint-related column.
owner	character varying(64)	Constraint creator.
position	smallint	Position of a column in a table.

12.3.12.46 DB_CONSTRAINTS

DB_CONSTRAINTS displays information about constraints accessible to the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-271 DB_CONSTRAINTS columns

Name	Type	Description
constraint_name	character varying(64)	Constraint name.
constraint_type	text	Constraint type. • **c**: check constraint • **f**: foreign key constraint • **p**: primary key constraint • **u**: unique constraint
table_name	character varying(64)	Name of a constraint-related table.
index_owner	character varying(64)	Owner of a constraint-related index (only for the unique constraint and primary key constraint).
index_name	character varying(64)	Name of a constraint-related index (only for the unique constraint and primary key constraint).
owner	character varying(64)	Constraint creator.

12.3.12.47 DB_DEPENDENCIES

DB_DEPENDENCIES displays the dependency relationship between types, tables, views, stored procedures, functions, and triggers accessible to the current user. It is accessible to all users. This view exists in the PG_CATALOG and SYS schemas.

Table 12-272 DB_DEPENDENCIES columns

Name	Type	Description
owner	name	Object owner.
name	name	Object name.
type	character varying(18)	Object class.
referenced_owner	name	Owner of the referenced object.
referenced_name	name	Name of the referenced object.
referenced_type	character varying(18)	Type of the referenced object.
referenced_link_name	character varying(128)	Not supported. Set it to **NULL**.
dependency_type	character varying(4)	Not supported. Set it to **NULL**.

12.3.12.48 DB_ERRORS

DB_ERRORS displays the latest compilation error information about storage objects accessible to users. By default, it is accessible to all users. This view exists in both PG_CATALOG and SYS schema.

Table 12-273 DB_ERRORS columns

Name	Type	Description
owner	character varying(128)	Object owner.
name	character varying(128)	Object name.
type	character varying(12)	Object type. • **PROCEDURE** • **FUNCTION** • **PACKAGE** • **PACKAGE BODY**
sequence	numeric	Serial number.
line	numeric	Row where an error occurs.
position	numeric	Not supported. Its value is **NULL**.
text	character varying(4000)	Error text.
attribute	character varying(9)	Attribute tag: ERROR.

Name	Type	Description
message_number	numeric	Not supported. Its value is **NULL**.

12.3.12.49 DB_IND_COLUMNS

DB_IND_COLUMNS displays all index columns accessible to the current user. This view exists in both PG_CATALOG and SYS schema.

Table 12-274 DB_IND_COLUMNS columns

Name	Type	Description
index_owner	character varying(64)	Index owner.
index_name	character varying(64)	Index name.
table_owner	character varying(64)	Table owner.
table_name	character varying(64)	Table name.
column_name	name	Column name.
column_position	smallint	Position of a column in an index.
column_length	numeric	Column length
char_length	numeric	Length of a column (character type).
descend	character varying(4)	Sorting mode of a column. The value can be **DESC** or **ASC**.
collated_column_id	numeric	Provides the internal serial number of the language sorting columns.

12.3.12.50 DB_IND_EXPRESSIONS

DB_IND_EXPRESSIONS displays information about expression indexes accessible to the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-275 DB_IND_EXPRESSIONS columns

Name	Type	Description
index_owner	character varying(64)	Index owner.

Name	Type	Description
index_name	character varying(64)	Index name.
table_owner	character varying(64)	Table owner.
table_name	character varying(64)	Table name.
column_expression	text	Function-based index expression of a specified column.
column_position	smallint	Position of a column in an index.

12.3.12.51 DB_INDEXES

DB_INDEXES displays information about indexes accessible to the current user. This view exists in both PG_CATALOG and SYS schema.

Table 12-276 DB_INDEXES columns

Name	Type	Description
owner	character varying(64)	Index owner.
index_name	character varying(64)	Index name.
table_name	character varying(64)	Name of a table corresponding to an index.
uniqueness	text	Specifies whether the index is a unique index. • **UNIQUE**: unique index. • **NONUNIQUE**: non-unique index.
partitioned	character(3)	Specifies whether the index has the property of partitioned tables. • **Yes**: The index has the property of a partitioned table. • **No**: The index does not have the property of a partitioned table.

Name	Type	Description
generated	character varying(1)	Specifies whether the index name is generated by the system. • **y**: The index name is generated by the system. • **n**: The index name is not generated by the system.
index_type	character varying(27)	Index type. • **NORMAL**: Index attributes are simple references, and the expression tree is empty. • **FUNCTION-BASED NORMAL**: Expression trees are used for index attributes that are not simple column references.
table_owner	character varying(128)	Owner of an index object.
table_type	character(11)	Type of an index object • **TABLE**: The index object is of the table type.
tablespace_name	character varying(30)	Name of the tablespace that contains the index.
status	character varying(8)	Status of a non-partitioned index. • **VALID**: Non-partitioned indexes can be used for query. • **UNUSABLE**: The non-partitioned index is unavailable. • **N/A**: The index has the property of a partitioned table.
compression	character varying(13)	Not supported. Its value is **NULL**.

Name	Type	Description
prefix_length	numeric	Not supported. Its value is **NULL**.
ini_trans	numeric	Not supported. Its value is **NULL**.
max_trans	numeric	Not supported. Its value is **NULL**.
initial_extent	numeric	Not supported. Its value is **NULL**.
next_extent	numeric	Not supported. Its value is **NULL**.
min_extents	numeric	Not supported. Its value is **NULL**.
max_extents	numeric	Not supported. Its value is **NULL**.
pct_increase	numeric	Not supported. Its value is **NULL**.
pct_threshold	numeric	Not supported. Its value is **NULL**.
include_column	numeric	Not supported. Its value is **NULL**.
freelists	numeric	Not supported. Its value is **NULL**.
freelist_groups	numeric	Not supported. Its value is **NULL**.
pct_free	numeric	Not supported. Its value is **NULL**.
logging	character varying(3)	Not supported. Its value is **NULL**.
blevel	numeric	Not supported. Its value is **NULL**.
leaf_blocks	numeric	Not supported. Its value is **NULL**.
distinct_keys	numeric	Not supported. Its value is **NULL**.
avg_leaf_blocks_per_key	numeric	Not supported. Its value is **NULL**.
avg_data_blocks_per_key	numeric	Not supported. Its value is **NULL**.

Name	Type	Description
clustering_factor	numeric	Not supported. Its value is **NULL**.
num_rows	numeric	Not supported. Its value is **NULL**.
sample_size	numeric	Not supported. Its value is **NULL**.
last_analyzed	timestamp(0) without time zone	Not supported. Its value is **NULL**.
degree	character varying(40)	Not supported. Its value is **NULL**.
instances	character varying(40)	Not supported. Its value is **NULL**.
temporary	character varying(1)	Not supported. Its value is **NULL**.
secondary	character varying(1)	Not supported. Its value is **NULL**.
buffer_pool	character varying(7)	Not supported. Its value is **NULL**.
flash_cache	character varying(7)	Not supported. Its value is **NULL**.
cell_flash_cache	character varying(7)	Not supported. Its value is **NULL**.
user_stats	character varying(3)	Not supported. Its value is **NULL**.
duration	character varying(15)	Not supported. Its value is **NULL**.
pct_direct_access	numeric	Not supported. Its value is **NULL**.
ityp_owner	character varying(128)	Not supported. Its value is **NULL**.
ityp_name	character varying(128)	Not supported. Its value is **NULL**.
parameters	character varying(1000)	Not supported. Its value is **NULL**.
global_stats	character varying(3)	Not supported. Its value is **NULL**.
domidx_status	character varying(12)	Not supported. Its value is **NULL**.

Name	Type	Description
domidx_opstatus	character varying(6)	Not supported. Its value is **NULL**.
funcidx_status	character varying(8)	Not supported. Its value is **NULL**.
join_index	character varying(3)	Not supported. Its value is **NULL**.
iot_redundant_pkey_elim	character varying(3)	Not supported. Its value is **NULL**.
dropped	character varying(3)	Not supported. Its value is **NULL**.
visibility	character varying(9)	Specifies whether the index is visible to the optimizer. • **VISIBLE**: The index is visible to the optimizer. • **INVISIBLE**: The index is invisible to the optimizer.
domidx_management	character varying(14)	Not supported. Its value is **NULL**.
segment_created	character varying(3)	Not supported. Its value is **NULL**.
orphaned_entries	character varying(3)	Not supported. Its value is **NULL**.
indexing	character varying(7)	Not supported. Its value is **NULL**.
auto	character varying(3)	Not supported. Its value is **NULL**.

12.3.12.52 DB_OBJECTS

DB_OBJECTS displays all database objects accessible to the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-277 DB_OBJECTS columns

Name	Type	Description
owner	name	Object owner.

Name	Type	Description
object_name	name	Object name.
object_id	oid	Object OID.
object_type	name	Object class.
namespace	oid	ID of the namespace where an object resides.
temporary	character(1)	Specifies whether an object is a temporary object.
status	character varying(7)	Object status.
subobject_name	name	Subobject name of an object.
generated	character(1)	Specifies whether an object name is generated by the system.
created	timestamp with time zone	Creation time of an object.
last_ddl_time	timestamp with time zone	Last modification time of an object.
default_collation	character varying(100)	Default collation of objects.
data_object_id	numeric	Not supported. Set it to **NULL**.
timestamp	character varying(19)	Not supported. Set it to **NULL**.
secondary	character varying(1)	Not supported. Set it to **NULL**.
edition_name	character varying(128)	Not supported. Set it to **NULL**.
sharing	character varying(18)	Not supported. Set it to **NULL**.
editionable	character varying(1)	Not supported. Set it to **NULL**.
oracle_maintained	character varying(1)	Not supported. Set it to **NULL**.
application	character varying(1)	Not supported. Set it to **NULL**.
duplicated	character varying(1)	Not supported. Set it to **NULL**.
sharded	character varying(1)	Not supported. Set it to **NULL**.

Name	Type	Description
created_appid	numeric	Not supported. Set it to **NULL**.
modified_appid	numeric	Not supported. Set it to **NULL**.
created_vsnid	numeric	Not supported. Set it to **NULL**.
modified_vsnid	numeric	Not supported. Set it to **NULL**.

NOTICE

For details about the value ranges of **created** and **last_ddl_time**, see **PG_OBJECT**.

12.3.12.53 DB_PROCEDURES

DB_PROCEDURES displays information about all stored procedures or functions accessible to the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-278 DB_PROCEDURES columns

Name	Type	Description
owner	name	Object owner.
object_name	name	Object name.

12.3.12.54 DB_SCHEDULER_JOB_ARGS

DB_SCHEDULER_JOB_ARG displays the parameters related to the tasks accessible to the current user. This view is accessible to all users and displays all information accessible to the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-279 DB_SCHEDULER_JOB_ARGS columns

Name	Type	Description
owner	character varying(128)	Owner of the job to which the parameter belongs.
job_name	character varying(128)	Name of the job to which the parameter belongs.
argument_name	character varying(128)	Parameter name.
argument_position	numeric	Position of the parameter in the parameter list.

Name	Type	Description
argument_type	character varying(257)	Data type of a parameter.
value	character varying(4000)	Parameter value.
anydata_value	character varying(4000)	Not supported. Set it to **NULL**.
out_argument	character varying(5)	Reserved column. Set it to **NULL**.

12.3.12.55 DB_SCHEDULER_PROGRAM_ARGS

DB_SCHEDULER_PROGRAM_ARG displays the parameters related to the programs accessible to the current user. This view is accessible to all users and displays all information accessible to the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-280 DB_SCHEDULER_PROGRAM_ARGS columns

Name	Type	Description
owner	character varying(128)	Owner of the program to which the parameter belongs.
program_name	character varying(128)	Name of the program to which the parameter belongs.
argument_name	character varying(128)	Parameter name.
argument_position	numeric	Position of the parameter in the parameter list.
argument_type	character varying(257)	Data type of a parameter.
metadata_attribute	character varying(19)	Not supported. Set it to **NULL**.
default_value	character varying(4000)	Default parameter value.
anydata_default_value	character varying(4000)	Not supported. Set it to **NULL**.
out_argument	character varying(5)	Reserved column. Set it to **NULL**.

12.3.12.56 DB_SEQUENCES

DB_SEQUENCES displays all sequences accessible to the current user. This view exists in both PG_CATALOG and SYS schema.

Table 12-281 DB_SEQUENCES columns

Name	Type	Description
sequence_owner	name	Owner of a sequence.
sequence_name	name	Name of the sequence.
min_value	int16	Minimum value of the sequence.
max_value	int16	Maximum value of the sequence.
increment_by	int16	Value by which the sequence is incremented.
cycle_flag	character(1)	Specifies whether a sequence is a cyclic sequence. Value range: • **Y**: It is a cyclic sequence. • **N**: It is not a cyclic sequence.
order_flag	character varying(1)	Specifies whether a sequence occurs in a request sequence. This parameter is not supported. Set it to **NULL**.
cache_size	int16	Size of the sequence disk cache.
last_number	int16	Value of the previous sequence.
scale_flag	character varying(1)	Specifies whether a sequence is a scalable sequence. Not supported. Its value is **NULL**.
extend_flag	character varying(1)	Specifies whether the value generated by a scalable sequence exceeds the maximum or minimum value of the sequence. Not supported. Its value is **NULL**.
sharded_flag	character varying(1)	Specifies whether a sequence is a shard sequence. Not supported. Its value is **NULL**.
session_flag	character varying(1)	Specifies whether a sequence has a private session. Not supported. Its value is **NULL**.
keep_value	character varying(1)	Specifies whether to retain the sequence value during replay after a failure. This parameter is not supported. Set it to **NULL**.

12.3.12.57 DB_SOURCE

DB_SOURCE displays the definition information about stored procedures, functions, triggers, and packages accessible to the current user. This view exists in both PG_CATALOG and SYS schema.

Table 12-282 DB_SOURCE columns

Name	Type	Description
owner	name	Object owner.
name	name	Object name.
type	name	Object type. Its values can be **function**, **package**, **package body**, **procedure**, and **trigger**.
line	numeric	Number of the line in the definition information.
text	text	Text source of the storage object.
origin_con_id	character varying(256)	Not supported. Its value is **0**.

12.3.12.58 DB_SYNONYMS

DB_SYNONYMS displays all synonyms accessible to the current user.

Table 12-283 DB_SYNONYMS columns

Name	Type	Description
owner	text	Owner of a synonym. The owner of the PUBLIC synonym is PUBLIC.
schema_name	text	Name of the schema to which the synonym belongs. The name of the schema to which the PUBLIC synonym belongs is NULL.
synonym_name	text	Synonym name.
table_owner	text	Owner of the associated object. Although the column is called **table_owner**, its associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.

Name	Type	Description
table_name	text	Name of the associated object. Although the column is called **table_name**, its associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.
table_schema_name	text	Schema name of the associated object. Although the column is called **table_schema_name**, its associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.
db_link	character varying(128)	Reserved column. Its value is **NULL**.
origin_con_id	character varying(256)	Not supported. Its value is **0**.

12.3.12.59 DB_TAB_COL_STATISTICS

DB_TAB_COL_STATISTICS displays column statistics and histogram information extracted from DB_TAB_COLUMNS. All users can access this view. This view exists in both PG_CATALOG and SYS schema. The values of the **LOW_VALUE** and **HIGH_VALUE** columns in this view are different from those in database A due to different underlying table structures. When the value of **LOW_VALUE** is a high-frequency value, the value of **LOW_VALUE** in GaussDB is the second smallest value. When **HIGH_VALUE** is a high-frequency value, **HIGH_VALUE** of GaussDB is the second highest value. The value of the **HISTOGRAM** column is different from that of database A due to different statistical methods. GaussDB supports only two types of histograms: frequency and equi-width. The value of the **SCOPE** column in GaussDB is different from that in database A because GaussDB does not support global temporary table statistics. GaussDB database supports only local temporary table statistics, and the default value is **SHARED**.

Table 12-284 DB_TAB_COL_STATISTICS columns

Name	Type	Description
owner	character varying(128)	Table owner.
table_name	character varying(128)	Table name.
column_name	character varying(128)	Column name.
num_distinct	numeric	Number of different values in a column.
low_value	raw	Low value in a column.

Name	Type	Description
high_value	raw	High value in a column.
density	numeric	• If there is a histogram on **COLUMN_NAME**, this column displays the selectivity of values in the histogram that span less than two endpoints. It does not represent the selectivity of values that span two or more endpoints. • If no histogram is available on **COLUMN_NAME**, the value of this column is **1/NUM_DISTINCT**.
num_nulls	numeric	Number of empty values in a column.
num_buckets	numeric	Number of buckets in the histogram of a column.
sample_size	numeric	Sample size used to analyze a column.
last_analyzed	timestamp(0) without time zone	Date when a column was last analyzed. After the database is restarted, data loss will occur.
global_stats	character varying(3)	Not supported. Its value is **NULL**.
user_stats	character varying(3)	Not supported. Its value is **NULL**.
notes	character varying(99)	Not supported. Its value is **NULL**.
avg_col_len	numeric	Average length of a column, in bytes.
histogram	character varying(15)	Specifies whether the histogram exists and the type of the histogram. • **NONE**: no histogram. • **FREQUENCY**: frequency histogram. • **EQUI-WIDTH**: equal-width histogram.
scope	character varying(7)	For statistics collected on any table other than global temporary tables, the value is SHARED (indicating that the statistics are shared among all sessions).
schema	character varying(64)	Name of the namespace to which the column belongs.

12.3.12.60 DB_TAB_COLUMNS

DB_TAB_COLUMNS displays description information about columns of tables and views accessible to the current user. This view exists in both PG_CATALOG and SYS schema. This view is accessible to all users and displays all information accessible to the current user.

Table 12-285 DB_TAB_COLUMNS columns

Name	Type	Description
owner	character varying(64)	Table owner.
table_name	character varying(64)	Table name.
column_name	character varying(64)	Column name.
data_type	character varying(128)	Data type of a column.
data_type_mod	character varying(3)	Not supported. Its value is **NULL**.
data_type_owner	character varying(128)	Owner of the data type of a column.
data_length	integer	Length of a column, in bytes.
data_precision	integer	Precision of a data type. This column is valid for the numeric data type and **NULL** for other data types.
data_scale	integer	Number of decimal places. This column is valid for the numeric data type and is set to **0** for other data types.
nullable	bpchar	Specifies whether a column can be empty (**n** for the primary key constraint and non-null constraint). Value range: • **y**: yes. • **n**: no.
column_id	integer	Sequence number of a column when a table is created.
default_length	numeric	Length of the default value of a column, in bytes.
data_default	text	Default value of a column.

Name	Type	Description
num_distinct	numeric	Number of different values in a column.
low_value	raw	Minimum value in a column.
high_value	raw	Maximum value in a column.
density	numeric	Column density.
num_nulls	numeric	Number of empty values in a column.
num_buckets	numeric	Number of buckets in the histogram of a column.
last_analyzed	timestamp(0) without time zone	Last analysis date.
sample_size	numeric	Sample size used to analyze a column.
character_set_name	character varying(44)	Not supported. Its value is **NULL**.
char_col_decl_length	numeric	Declaration length of a column of the character type.
global_stats	character varying(3)	Not supported. Its value is **NO**.
user_stats	character varying(3)	Not supported. Its value is **NO**.
avg_col_len	numeric	Average length of a column, in bytes.
char_length	numeric	Column length (in bytes) which is valid only for the varchar, nvarchar2, bpchar, and char types.
char_used	character varying(1)	Not supported. Set it to **B** if the data type is varchar, nvarchar2, bpchar, or char. For other data types, set it to **NULL**.
v80_fmt_image	character varying(3)	Not supported. Its value is **NULL**.
data_upgraded	character varying(3)	Not supported. Its value is **YES**.
histogram	character varying(15)	Specifies whether the histogram exists and the type of the histogram. • **NONE**: no histogram. • **FREQUENCY**: frequency histogram. • **EQUI_WIDTH**: equal-width histogram.

Name	Type	Description
default_on_null	character varying(3)	Not supported. Its value is **NULL**.
identity_column	character varying(3)	Not supported. Its value is **NULL**.
evaluation_edition	character varying(128)	Not supported. Its value is **NULL**.
unusable_before	character varying(128)	Not supported. Its value is **NULL**.
unusable_beginning	character varying(128)	Not supported. Its value is **NULL**.
collation	character varying(100)	Collation rule of a column. This column conflicts with reserved keywords. Therefore, add the view name when calling this column.
comments	text	Comments.
schema	character varying(64)	Name of the namespace to which the column belongs.

12.3.12.61 DB_TAB_COMMENTS

DB_TAB_COMMENTS displays comments about all tables and views accessible to the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-286 DB_TAB_COMMENTS columns

Name	Type	Description
owner	character varying(128)	Owner of a table or view.
table_name	character varying(128)	Name of a table or view.
table_type	character varying(11)	Object type.
comments	text	Comments.
origin_con_id	numeric	Not supported. Set it to **0**.
schema	character varying(64)	Name of the namespace to which the table belongs.

12.3.12.62 DB_TAB_HISTOGRAMS

DB_TAB_HISTOGRAMS displays histogram statistics about the tables and views accessible to the current user, that is, the distribution of data in each column of

the table. All users can access this view. This view exists in the PG_CATALOG and SYS schemas.

Table 12-287 DB_TAB_HISTOGRAMS columns

Name	Type	Description
owner	character varying(128)	Table owner.
table_name	character varying(128)	Table name.
column_name	character varying(4000)	Column name or attribute of an object column.
endpoint_number	numeric	Bucket ID of the histogram.
endpoint_value	numeric	Not supported. Set it to **NULL**.
endpoint_actual_value	character varying(4000)	Actual value of the bucket endpoint.
endpoint_actual_value_raw	raw	Not supported. Set it to **NULL**.
endpoint_repeat_count	numeric	Not supported. Set it to **NULL**.
scope	character varying(7)	Not supported. Set it to **SHARED**.

12.3.12.63 DB_TAB_MODIFICATIONS

DB_TAB_MODIFICATIONS displays statistics about modifications to tables accessible to the current user since the last statistics collection on the tables. Currently, this view displays only the statistics data of tables on which INSERT, DELETE, and UPDATE operations have been performed. All users can access this view. This view exists in both PG_CATALOG and SYS schema.

Table 12-288 DB_TAB_MODIFICATIONS columns

Name	Type	Description
table_owner	character varying(128)	Table owner.
table_name	character varying(128)	Table name.
partition_name	character varying(128)	Partition name.

Name	Type	Description
subpartition_name	character varying(128)	Subpartition name.
inserts	numeric	Approximate number of insertions since the last statistics collection.
updates	numeric	Approximate number of updates since the last statistics collection.
deletes	numeric	Approximate number of deletions since the last statistics collection.
timestamp	timestamp(0) without time zone	Last modification time. Currently, the modification time of a partitioned table is not supported. The value is **NULL**.
truncated	character varying(3)	Not supported. Its value is **NULL**.
drop_segments	numeric	Not supported. Its value is **NULL**.
schema_name	character varying(128)	Name of the schema to which the table belongs.

12.3.12.64 DB_TAB_STATS_HISTORY

DB_TAB_STATS_HISTORY displays the tables, partitions, and subpartitions involved in table statistics and the time when table statistics are collected. All users can access this view. This view exists in both PG_CATALOG and SYS schema.

Table 12-289 DB_TAB_STATS_HISTORY columns

Name	Type	Description
owner	character varying(128)	Object owner.
table_name	character varying(128)	Table name.
partition_name	character varying(128)	Not supported. Its value is **NULL**.
subpartition_name	character varying(128)	Not supported. Its value is **NULL**.
stats_update_time	timestamp(6) with time zone	Time when statistics are updated. Database restart is not supported. Otherwise, data loss will occur.

12.3.12.65 DB_TABLES

DB_TABLES displays all tables accessible to the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-290 DB_TABLES columns

Name	Type	Description
owner	character varying(64)	Table owner
table_name	character varying(64)	Table name.
tablespace_name	character varying(64)	Tablespace name of the table.
dropped	character varying	Specifies whether the current table is deleted. • **YES**: It is deleted. • **NO**: It is not deleted.
num_rows	numeric	Estimated number of rows in the table.
status	character varying(8)	Specifies whether the current table is valid. • **VALID**: The current table is valid. • **UNUSABLE**: The current table is unavailable.
sample_size	numeric	Number of samples used for analyzing the table.
temporary	character(1)	Specifies whether the table is a temporary table. • **Y**: The table is a temporary table. • **N**: The table is not a temporary table.
pct_free	numeric	Minimum percentage of free space in a block.
ini_trans	numeric	Initial number of transactions.
max_trans	numeric	Maximum number of transactions.
avg_row_len	integer	Average number of bytes in each row.

Name	Type	Description
partitioned	character varying(3)	Specifies whether a table is a partitioned table. • **YES**: The table is a partitioned table. • **NO**: The table is not a partitioned table.
last_analyzed	timestamp with time zone	Last time when the table was analyzed. Database restart is not supported. Otherwise, data loss will occur.
row_movement	character varying(8)	Specifies whether to allow partition row movement. • **ENABLED**: The partition row movement is allowed. • **DISABLED**: The partition row movement is not allowed.
compression	character varying(8)	Specifies whether to enable a table compression. • **ENABLED**: A table compression is enabled. • **DISABLED**: A table compression is disabled.
duration	character varying(15)	Time elapsed when a temporary table is processed. • **NULL**: The table is not a temporary table. • **sys$session**: The table is a temporary session table. • **sys$transaction**: The table is a temporary transaction table.
logical_replication	character varying(8)	Specifies whether logical replication is enabled for a table. • **ENABLED**: Logical replication is enabled. • **DISABLED**: Logical replication is disabled.

Name	Type	Description
external	character varying(3)	Specifies whether the table is an external table. • **YES**: The table is an external table. • **NO**: The table is not an external table.
logging	character varying(3)	Specifies whether to record logs for table changes. • **YES**: Logs are recorded for table changes. • **NO**: Logs are not recorded for table changes.
default_collation	character varying(100)	Default collation of a table.
degree	character varying(10)	Number of instances in a scanned table.
table_lock	character varying(8)	Specifies whether to enable a table lock. • **ENABLED**: The table lock is enabled. • **DISABLED**: The table lock is disabled.
nested	character varying(3)	Specifies whether a table is a nested table. • **YES**: The table is a nested table. • **NO**: The table is not a nested table.
buffer_pool	character varying(7)	Default buffer pool of a table.
flash_cache	character varying(7)	Not supported. The value is **NULL**.
cell_flash_cache	character varying(7)	Not supported. The value is **NULL**.
skip_corrupt	character varying(8)	Specifies whether to skip corrupted blocks during table scanning. • **ENABLED**: The corrupted block is skipped. • **DISABLED**: The corrupted block is not skipped.

Name	Type	Description
has_identity	character varying(3)	Specifies whether a table has an identifier column. • **YES**: There is an identifier column. • **NO**: There is no identifier column.
segment_created	character varying(3)	Specifies whether a table segment has been created. • **YES**: The table segment has been created. • **NO**: The table segment is not created.
monitoring	character varying(3)	Specifies whether to monitor the modification of a table. • **YES**: The modification of the table is monitored. • **NO**: The modification of the table is not monitored.
cluster_name	character varying(128)	Not supported. The value is **NULL**.
iot_name	character varying(128)	Not supported. The value is **NULL**.
pct_used	numeric	Not supported. The value is **NULL**.
initial_extent	numeric	Not supported. The value is **NULL**.
next_extent	numeric	Not supported. The value is **NULL**.
min_extents	numeric	Not supported. The value is **NULL**.
max_extents	numeric	Not supported. The value is **NULL**.
pct_increase	numeric	Not supported. The value is **NULL**.
freelists	numeric	Not supported. The value is **NULL**.
freelist_groups	numeric	Not supported. The value is **NULL**.

Name	Type	Description
backed_up	character varying(1)	Not supported. The value is **NULL**.
blocks	numeric	Not supported. The value is **NULL**.
empty_blocks	numeric	Not supported. The value is **NULL**.
avg_space	numeric	Not supported. The value is **NULL**.
chain_cnt	numeric	Not supported. The value is **NULL**.
avg_space_freelist_blocks	numeric	Not supported. The value is **NULL**.
num_freelist_blocks	numeric	Not supported. The value is **NULL**.
instances	character varying(10)	Not supported. The value is **NULL**.
cache	character varying(5)	Not supported. The value is **NULL**.
iot_type	character varying(12)	Not supported. The value is **NULL**.
secondary	character varying(1)	Not supported. The value is **NULL**.
global_stats	character varying(3)	Not supported. The value is **NULL**.
user_stats	character varying(3)	Not supported. The value is **NULL**.
cluster_owner	character varying(30)	Not supported. The value is **NULL**.
dependencies	character varying(8)	Not supported. The value is **NULL**.
compression_for	character varying(30)	Not supported. The value is **NULL**.
read_only	character varying(3)	Not supported. The value is **NULL**.
result_cache	character varying(7)	Not supported. The value is **NULL**.
clustering	character varying(3)	Not supported. The value is **NULL**.

Name	Type	Description
activity_tracking	character varying(23)	Not supported. The value is **NULL**.
dml_timestamp	character varying(25)	Not supported. The value is **NULL**.
container_data	character varying(3)	Not supported. The value is **NULL**.
inmemory_priority	character varying(8)	Not supported. The value is **NULL**.
inmemory_distribute	character varying(15)	Not supported. The value is **NULL**.
inmemory_compression	character varying(17)	Not supported. The value is **NULL**.
inmemory_duplicate	character varying(13)	Not supported. The value is **NULL**.
duplicated	character varying(1)	Not supported. The value is **NULL**.
sharded	character varying(1)	Not supported. The value is **NULL**.
hybrid	character varying(3)	Not supported. The value is **NULL**.
cellmemory	character varying(24)	Not supported. The value is **NULL**.
containers_default	character varying(3)	Not supported. The value is **NULL**.
container_map	character varying(3)	Not supported. The value is **NULL**.
extended_data_link	character varying(3)	Not supported. The value is **NULL**.
extended_data_link_map	character varying(3)	Not supported. The value is **NULL**.
inmemory_service	character varying(12)	Not supported. The value is **NULL**.
inmemory_service_name	character varying(1000)	Not supported. The value is **NULL**.
container_map_object	character varying(3)	Not supported. The value is **NULL**.
memoptimize_read	character varying(8)	Not supported. The value is **NULL**.

Name	Type	Description
memoptimize_write	character varying(8)	Not supported. The value is **NULL**.
has_sensitive_column	character varying(3)	Not supported. The value is **NULL**.
admit_null	character varying(3)	Not supported. The value is **NULL**.
data_link_dml_enabled	character varying(3)	Not supported. The value is **NULL**.
object_id_type	character varying(16)	Not supported. The value is **NULL**.
table_type_owner	character varying(128)	Not supported. The value is **NULL**.
table_type	character varying(128)	Not supported. The value is **NULL**.
compress_for	character varying(30)	Not supported. The value is **NULL**.

12.3.12.66 DB_TRIGGERS

DB_TRIGGERS displays information about triggers accessible to the current user. This view exists in both PG_CATALOG and SYS schema.

Table 12-291 DB_TRIGGERS columns

Name	Type	Description
trigger_name	character varying(64)	Trigger name.
table_name	character varying(64)	Relational table name.
table_owner	character varying(64)	Role name.
status	character varying(64)	Trigger status: • **O**: The trigger is enabled in origin or local mode. • **D**: The trigger is disabled. • **R**: The trigger is enabled in replica mode. • **A**: The trigger is always enabled.

12.3.12.67 DB_TYPES

DB_TYPES displays all object types accessible to the current user. All users can access this view. This view exists in the PG_CATALOG and SYS schemas.

Table 12-292 DB_TYPES columns

Name	Type	Description
owner	character varying(128)	Owner of the type.
type_name	character varying(128)	Type name.
type_oid	raw	Type OID.
typecode	character varying(128)	Type code.
attributes	numeric	Number of attributes in a type.
methods	numeric	Not supported. Set it to **0**.
predefined	character varying(3)	Specifies whether the type is a predefined type.
incomplete	character varying(3)	Specifies whether the type is an incomplete type.
final	character varying(3)	Not supported. Set it to **NULL**.
instantiable	character varying(3)	Not supported. Set it to **NULL**.
persistable	character varying(3)	Not supported. Set it to **NULL**.
supertype_owner	character varying(128)	Not supported. Set it to **NULL**.
supertype_name	character varying(128)	Not supported. Set it to **NULL**.
local_attributes	numeric	Not supported. Set it to **NULL**.
local_methods	numeric	Not supported. Set it to **NULL**.
typeid	raw	Not supported. Set it to **NULL**.

12.3.12.68 DB_VIEWS

DB_VIEWS displays the description about all views accessible to the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-293 DB_VIEWS columns

Name	Type	Description
owner	name	Owner of a view.
view_name	name	Name of a view.
text	text	Text of a view.
text_length	integer	Text length of a view.
text_vc	character varying(4000)	View creation statement. This column may truncate the view text. The BEQUEATH clause will not appear as part of the **text_vc** column in this view.
type_text_length	numeric	Not supported. Set it to **NULL**.
type_text	character varying(4000)	Not supported. Set it to **NULL**.
oid_text_length	numeric	Not supported. Set it to **NULL**.
oid_text	character varying(4000)	Not supported. Set it to **NULL**.
view_type_owner	character varying(128)	Not supported. Set it to **NULL**.
view_type	character varying(128)	Not supported. Set it to **NULL**.
superview_name	character varying(128)	Not supported. Set it to **NULL**.
editioning_view	character varying(1)	Not supported. Set it to **NULL**.
read_only	character varying(1)	Not supported. Set it to **NULL**.
container_data	character varying(1)	Not supported. Set it to **NULL**.
bequeath	character varying(12)	Not supported. Set it to **NULL**.
origin_con_id	character varying(256)	Not supported. Set it to **NULL**.
default_collation	character varying(100)	Not supported. Set it to **NULL**.
containers_default	character varying(3)	Not supported. Set it to **NULL**.

Name	Type	Description
container_map	character varying(3)	Not supported. Set it to **NULL**.
extended_data_link	character varying(3)	Not supported. Set it to **NULL**.
extended_data_link_map	character varying(3)	Not supported. Set it to **NULL**.
has_sensitive_column	character varying(3)	Not supported. Set it to **NULL**.
admit_null	character varying(3)	Not supported. Set it to **NULL**.
pdb_local_only	character varying(3)	Not supported. Set it to **NULL**.

12.3.12.69 DICT

DICT displays the description of data dictionary tables and system views in the database. This view exists in the PG_CATALOG and SYS schemas and all users can access this view.

Table 12-294 DICT columns

Name	Type	Description
table_name	character varying(64)	Object name.
comments	character varying(4000)	Comment on an object.

12.3.12.70 DICTIONARY

DICTIONARY displays the description of data dictionary tables and system views in the database. This view exists in the PG_CATALOG and SYS schemas and all users can access this view.

Table 12-295 DICTIONARY columns

Name	Type	Description
table_name	character varying(64)	Object name.

Name	Type	Description
comments	character varying(4000)	Comment on an object.

12.3.12.71 DUAL

DUAL is automatically created by the database based on the data dictionary and is used to save expression calculation results. It is accessible to all users. This view exists in the PG_CATALOG and SYS schemas.

Table 12-296 DUAL columns

Name	Type	Description
DUMMY	text	Expression calculation result.

12.3.12.72 DV_SESSIONS

DV_SESSIONS displays information about all active backend threads. By default, only the system administrator can access this view. Common users can access the view only after being authorized.

Table 12-297 DV_SESSIONS columns

Name	Type	Description
sid	bigint	OID of the active backend thread of the current session.
serial#	integer	Sequence number of the backend thread of the current activity, which is **0** in GaussDB.
user#	oid	OID of the user that has logged in to the backend thread. The OID is **0** if the backend thread is a global auxiliary thread.
username	name	Name of the user logged in to the backend process. **username** is null if the backend thread is a global auxiliary thread. **application_name** can be identified by associating with **pg_stat_get_activity()**. Example: `SELECT s.*,a.application_name FROM DV_SESSIONS AS s LEFT JOIN pg_stat_get_activity(NULL) AS a ON s.sid=a.sessionid;`

12.3.12.73 DV_SESSION_LONGOPS

DV_SESSION_LONGOPS displays the progress of ongoing operations. Users can access this view only after being authorized.

Table 12-298 DV_SESSION_LONGOPS columns

Name	Type	Description
sid	bigint	OID of the running backend thread.
serial#	integer	Sequence number of the running backend thread, which is **0** in GaussDB.
sofar	integer	Completed workload, which is null in GaussDB.
totalwork	integer	Total workload, which is null in GaussDB.

12.3.12.74 GS_ALL_CONTROL_GROUP_INFO

GS_ALL_CONTROL_GROUP_INFO displays all Cgroup information in a database.

Table 12-299 GS_ALL_CONTROL_GROUP_INFO columns

Name	Type	Description
name	text	Cgroup name.
type	text	Cgroup type. • **GROUP_NONE**: no group. • **GROUP_TOP**: top group. • **GROUP_CLASS**: Class Cgroup, which does not control any thread. • **GROUP_BAKWD**: backend thread Cgroup. • **GROUP_DEFWD**: default Cgroup, which controls only the query threads at this level. • **GROUP_TSWD**: time-sharing Cgroup of each user, which controls the query threads at the bottom layer.
gid	bigint	Cgroup ID.
classgid	bigint	ID of the class Cgroup where a workload Cgroup belongs.
class	text	Class Cgroup.
workload	text	Workload Cgroup.
shares	bigint	CPU quota allocated to the Cgroup.
limits	bigint	Limit of CPU resources allocated to a Cgroup.
wdlevel	bigint	Workload Cgroup level.

Name	Type	Description
cpucores	text	Information about the CPU cores used by a Cgroup.

12.3.12.75 GS_ALL_PREPARED_STATEMENTS

GS_ALL_PREPARED_STATEMENTS displays information about prepared statements that are available in all sessions. This view is accessible only to system administrators.

Table 12-300 GS_ALL_PREPARED_STATEMENTS columns

Name	Type	Description
pid	bigint	Backend thread ID. **NOTE** In thread pool mode, **pid** indicates the ID of the thread bound to the current session. When a session is executed on different threads, **pid** changes accordingly. In thread pool mode, statements are associated with **sessionid** but not **pid**. You are advised to use **sessionid** for associated query.
sessionid	bigint	Current session ID.
global_sessionid	text	Global session ID.
name	text	Prepared statement identifier.
statement	text	Query string for creating this prepared statement. ● The column is the PREPARE statements submitted by the client for prepared statements created from SQL. ● The column is the text of the prepared statements for prepared statements created through a front-end/back-end protocol.
prepare_time	timestamp with time zone	Timestamp when the prepared statement is created.
parameter_types	regtype[]	Expected parameter type of the prepared statement in the form of a regtype array. The OID corresponding to this array can be obtained by converting the regtype value.
from_sql	boolean	● **True** if the prepared statement was created through the PREPARE statement. ● **False** if the statement was prepared through the frontend/backend protocol.

12.3.12.76 GS_COMM_PROXY_THREAD_STATUS

GS_COMM_PROXY_THREAD_STATUS displays statistics on packets sent and received by the proxy communication library comm_proxy. This view displays statistics about data sent and received by comm_proxy only when the user-mode network deployment mode is enabled during the installation of the centralized database and **enable_dfx** of **comm_proxy_attr** is set to **true**. In other scenarios, this view cannot be queried.

Table 12-301 GS_COMM_PROXY_THREAD_STATUS columns

Name	Type	Description
ProxyThreadId	bigint	ID of the current network proxy thread comm_proxy.
ProxyCpuAffinity	text	NUMA-CPU affinity of the current network proxy thread comm_proxy, indicating the NUMA and CPU ID.
ThreadStartTime	text	Start time of the current network proxy thread comm_proxy.
RxPckNums	bigint	Number of packets received by the current network proxy thread comm_proxy.
TxPckNums	bigint	Number of packets sent by the current network proxy thread comm_proxy.
RxPcks	double precision	Number of packets received by the network proxy thread comm_proxy per second.
TxPcks	double precision	Number of packets sent by the network proxy thread comm_proxy per second.

12.3.12.77 GS_FILE_STAT

GS_FILE_STAT records statistics about data file I/Os to indicate I/O performance and detect performance problems such as abnormal I/O operations.

Table 12-302 GS_FILE_STAT columns

Name	Type	Description
filenum	oid	File identifier
dbid	oid	Database ID
spcid	oid	Tablespace ID

Name	Type	Description
phyrds	bigint	Number of times of reading physical files
phywrts	bigint	Number of times of writing into physical files
phyblkrd	bigint	Number of times of reading physical file blocks
phyblkwrt	bigint	Number of times of writing into physical file blocks
readtim	bigint	Total duration of reading, in microseconds
writetim	bigint	Total duration of writing, in microseconds
avgiotim	bigint	Average duration of reading and writing, in microseconds
lstiotim	bigint	Duration of the last file reading, in microseconds
miniotim	bigint	Minimum duration of reading and writing, in microseconds
maxiowtm	bigint	Maximum duration of reading and writing, in microseconds

12.3.12.78 GS_GET_CONTROL_GROUP_INFO

This view is not supported in the centralized system.

12.3.12.79 GS_GLC_MEMORY_DETAIL

GS_GLC_MEMORY_DETAIL displays the memory usage of the global PL/pgSQL cache in all databases. This view is available only when **enable_global_plsqlcache** is set to **on**.

Table 12-303 GS_GLC_MEMORY_DETAIL columns

Name	Type	Description
contextname	text	Name of a memory object.
database	text	Database to which the memory object belongs. **"pkg_bucket"** and **"func_bucket"** are displayed as **"NULL"**.
schema	text	Database to which the memory object belongs. **"pkg_bucket"** and **"func_bucket"** are displayed as **"NULL"**.

Name	Type	Description
type	text	Object type. • **"pkg_bucket"**: indicates that the object is the parent node of a package object. • **"func_bucket"**: indicates that the object is the parent node of a function or stored procedure. • **"pkg"**: indicates that the object is a package object. • **"func"**: indicates that the object is a function or stored procedure.
status	text	Current status of the cache object. **"valid"** indicates that the cache object is available, and **'invalid'** indicates that the cache object is unavailable. The **"pkg_bucket"** and **"func_bucket"** objects have no status and are displayed as **"NULL"**.
location	text	Current location of a cached object. It is displayed as **"in_global_hash_table"** in the cache hash table and **"in_global_expired_list"** in the invalid linked list. The **"pkg_bucket"** and **"func_bucket"** objects are displayed as **"NULL"**.
env	bigint	Environment parameter when an object is created, that is, the value of **behavior_compat_flags**. The **"pkg_bucket"** and **"func_bucket"** objects are displayed as **0**.
usedsize	bigint	Size of a cached object.
usecount	bigint	Number of objects that are referencing the global cache. When no object references the global compilation product, the number is 0.

12.3.12.80 GS_GLOBAL_ARCHIVE_STATUS

GS_GLOBAL_ARCHIVE_STATUS describes the archiving progress of DNs and all shards, including the shard name (**node_name**), archiving location (**restart_lsn**), name of the primary or standby node (**archive_node**), and current log location (**current_xlog_location**). To query this view, you need to enable the archiving function of the database and query the view from the primary DN.

Table 12-304 GS_GLOBAL_ARCHIVE_STATUS columns

Name	Type	Description
node_name	text	Shard name.
restart_lsn	text	Archiving location.

Name	Type	Description
archive_node	text	Name of the primary or standby node where archiving is performed.
current_xlog_location	text	Current log position.

12.3.12.81 GS_GSC_MEMORY_DETAIL

GS_GSC_MEMORY_DETAIL displays the memory usage of the global system cache of the current process on the current node. The data is displayed only when GSC is enabled.

The query is separated by the database memory context. Therefore, some memory statistics are missing. The memory context corresponding to the missing memory statistics is **GlobalSysDBCache**.

Table 12-305 GS_GSC_MEMORY_DETAIL columns

Name	Type	Description
db_id	text	Database ID.
totalsize	numeric	Total size of the shared memory, in bytes.
freesize	numeric	Remaining size of the shared memory, in bytes.
usedsize	numeric	Used size of the shared memory, in bytes.

12.3.12.82 GS_INSTANCE_TIME

Records time consumption information of the current node. The time consumption information is classified into the following types:

- DB_TIME: effective time spent by jobs in multi-core scenarios.
- CPU_TIME: CPU time cost.
- EXECUTION_TIME: time spent in the executor.
- PARSE_TIME: time spent on parsing SQL statements.
- PLAN_TIME: time spent on generating plans.
- REWRITE_TIME: time spent on rewriting SQL statements.
- PL_EXECUTION_TIME: execution time of the PL/SQL stored procedure.
- PL_COMPILATION_TIME: compilation time of the PL/SQL stored procedure.
- NET_SEND_TIME: time spent on the network.
- DATA_IO_TIME: time spent on I/Os.

Table 12-306 GS_INSTANCE_TIME columns

Name	Type	Description
stat_id	integer	Statistics ID
stat_name	text	Type name
value	bigint	Time value, in μs

12.3.12.83 GS_LSC_MEMORY_DETAIL

GS_LSC_MEMORY_DETAIL displays statistics about the local SysCache memory usage of all threads based on the MemoryContext node. The statistics are available only when GSC is enabled.

Table 12-307 GS_LSC_MEMORY_DETAIL columns

Name	Type	Description
threadid	text	Thread start time + thread ID (string: *timestamp.sessionid*).
tid	bigint	Thread ID.
thrdtype	text	Thread type. It can be any thread type in the system, such as postgresql and wlmmonitor.
contextname	text	Name of the memory context.
level	smallint	Hierarchy of the memory context.
parent	text	Name of the parent memory context.
totalsize	bigint	Total size of the memory context, in bytes.
freesize	bigint	Total size of released memory in the current memory context, in bytes.
usedsize	bigint	Total size of used memory in the current memory context, in bytes.

12.3.12.84 GS_MY_PLAN_TRACE

GS_MY_PLAN_TRACE is a view of the GS_PLAN_TRACE system catalog. This view displays the plan traces of the current user.

Table 12-308 GS_MY_PLAN_TRACE columns

Name	Type	Description
query_id	text	Unique ID of the current request.

Name	Type	Description
query	text	SQL statement of the current request. The value of this field cannot exceed the value of **track_activity_query_size**.
unique_sql_id	bigint	Unique ID of the SQL statement of the current request.
plan	text	Query plan text corresponding to the SQL statement of the current request. The size of this field cannot exceed 10 KB.
plan_trace	text	Details about the query plan generation process corresponding to the SQL statement of the current request. The value of this field cannot exceed 300 MB.
modifydate	timestamp with time zone	Time when the current plan trace is updated (that is, time when the plan trace is created).

12.3.12.85 GS_OS_RUN_INFO

GS_OS_RUN_INFO displays the running status of the OS.

Table 12-309 GS_OS_RUN_INFO columns

Name	Type	Description
id	integer	ID.
name	text	Name of the OS running status.
value	numeric	Value of the OS running status
comments	text	Remarks of the OS running status.
cumulative	boolean	Specifies whether the value of the OS running status is cumulative.

12.3.12.86 GS_REDO_STAT

GS_REDO_STAT displays the log replay of session threads.

Table 12-310 GS_REDO_STAT columns

Name	Type	Description
phywrts	bigint	Number of times that data is written during log replay.

Name	Type	Description
phyblkwrt	bigint	Number of data blocks written during log replay.
writetim	bigint	Total time required for writing data during log replay.
avgiotim	bigint	Average time required for writing data during log replay.
lstiotim	bigint	Time consumed by the last data write operation during log replay.
miniotim	bigint	Minimum time consumed by a single data write operation during log replay.
maxiowtm	bigint	Maximum time consumed by a single data write operation during log replay.

12.3.12.87 GS_SESSION_ALL_SETTINGS

GS_SESSION_ALL_SETTINGS displays the full GUC parameter settings of all sessions on the local node. Only the **sysadmin** and **monadmin** users have the permission to view this view.

Table 12-311 GS_SESSION_ALL_SETTINGS columns

Name	Type	Description
sessionid	bigint	Session ID.
pid	bigint	Backend thread ID.
name	text	Parameter name.
setting	text	Current parameter value.
unit	text	Implicit unit of a parameter.

12.3.12.88 GS_SESSION_MEMORY

GS_SESSION_MEMORY displays the memory usage at the session level, including all the memory allocated to GaussDB and Stream threads on DNs for jobs currently executed by users. If the GUC parameter **enable_memory_limit** is set to **off**, this view is unavailable.

Table 12-312 GS_SESSION_MEMORY columns

Name	Type	Description
sessid	text	Thread start time and ID.

Name	Type	Description
init_mem	integer	Memory allocated to the currently executed jobs before they enter the executor, in MB.
used_mem	integer	Memory allocated to the currently executed jobs, in MB.
peak_mem	integer	Peak memory allocated to the currently executed jobs, in MB.

12.3.12.89 GS_SESSION_MEMORY_CONTEXT

GS_SESSION_MEMORY_CONTEXT displays the memory usage of all sessions based on the MemoryContext node. If the GUC parameter **enable_memory_limit** or **enable_thread_pool** is set to **off**, this view is unavailable.

The memory context **TempSmallContextGroup** collects information about all memory contexts whose value in the **totalsize** column is less than 8192 bytes in the current thread, and the number of the collected memory contexts is recorded in the **usedsize** column. Therefore, the **totalsize** and **freesize** columns for **TempSmallContextGroup** in the view display the corresponding information about all the memory contexts whose value in the **totalsize** column is less than 8192 bytes in the current thread, and the **usedsize** column displays the number of these memory contexts.

Table 12-313 GS_SESSION_MEMORY_CONTEXT columns

Name	Type	Description
sessid	text	Session start time + session ID (character string: *timestamp.sessionid*).
threadid	bigint	ID of the thread bound to a session (**-1** if no thread is bound).
contextname	text	Name of the memory context.
level	smallint	Hierarchy of the memory context.
parent	text	Name of the parent memory context.
totalsize	bigint	Total size of the memory context, in bytes.
freesize	bigint	Total size of released memory in the current memory context, in bytes.
usedsize	bigint	Size of used memory in the memory context, in bytes. For **TempSmallContextGroup**, this parameter specifies the number of collected memory contexts.

> **CAUTION**
>
> This view is an O&M view and is used to locate memory problems. Do not query this view concurrently. If you query this view concurrently, the waiting time for new connections increases as the number of concurrent connections increases. As a result, new connections cannot be connected for a long time.

12.3.12.90 GS_SESSION_MEMORY_DETAIL

GS_SESSION_MEMORY_DETAIL displays the thread memory usage based on the MemoryContext node. When **enable_thread_pool** is set to **on**, this view contains memory usage of all threads and sessions. If the GUC parameter **enable_memory_limit** is set to **off**, this view is unavailable.

The memory context **TempSmallContextGroup** collects information about all memory contexts whose value in the **totalsize** column is less than 8192 bytes in the current thread, and the number of the collected memory contexts is recorded in the **usedsize** column. Therefore, the **totalsize** and **freesize** columns for **TempSmallContextGroup** in the view display the corresponding information about all the memory contexts whose value in the **totalsize** column is less than 8192 bytes in the current thread, and the **usedsize** column displays the number of these memory contexts.

You can run the **SELECT * FROM gs_session_memctx_detail(***threadid***, '');** statement to record information about all memory contexts of a thread into the *threadid_timestamp***.log** file in the *$GAUSSLOG*/**gs_log/***${node_name}*/**dumpmem** directory. *threadid* can be obtained from **sessid** in the following table.

Table 12-314 GS_SESSION_MEMORY_DETAIL columns

Name	Type	Description
sessid	text	1. When the thread pool is disabled (**enable_thread_pool = off**), this column indicates the thread start time + session ID (string: **timestamp.sessionid**). 2. When the thread pool is enabled (**enable_thread_pool = on**): If the memory context is at the thread level, this column indicates the thread start time + thread ID (string: **timestamp.threadid**). If the memory context is at the session level, the column indicates the thread start time + session ID (string: **timestamp.sessionid**).
sesstype	text	Thread name
contextname	text	Name of the memory context
level	smallint	Hierarchy of the memory context
parent	text	Name of the parent memory context
totalsize	bigint	Total size of the memory context, in bytes

Name	Type	Description
freesize	bigint	Total size of released memory in the memory context, in bytes
usedsize	bigint	Size of used memory in the memory context, in bytes. For **TempSmallContextGroup**, this parameter specifies the number of collected memory contexts.

CAUTION

This view is an O&M view and is used to locate memory problems. Do not query this view concurrently. If you query this view concurrently, the waiting time for new connections increases as the number of concurrent connections increases. As a result, new connections cannot be connected for a long time.

12.3.12.91 GS_SESSION_STAT

GS_SESSION_STAT displays the session states based on session threads or the **AutoVacuum** thread.

Table 12-315 GS_SESSION_STAT columns

Name	Type	Description
sessid	text	Thread ID and start time.
statid	integer	ID of the statistics session.
statname	text	Name of the statistics session.
statunit	text	Unit of the statistics session.
value	bigint	Value of the statistics session.

12.3.12.92 GS_SESSION_TIME

GS_SESSION_TIME displays the running time of session threads and time consumed in each execution phase.

Table 12-316 GS_SESSION_TIME columns

Name	Type	Description
sessid	text	Thread ID and start time.
stat_id	integer	ID of the statistics session.
stat_name	text	Type name of the statistics session.

Name	Type	Description
value	bigint	Value of the statistics session.

12.3.12.93 GS_SHARED_MEMORY_DETAIL

SHARED_MEMORY_DETAIL queries the usage information about shared memory contexts on the current node.

Table 12-317 GS_SHARED_MEMORY_DETAIL columns

Name	Type	Description
contextname	text	Name of the memory context
level	smallint	Level of the memory context
parent	text	Name of the parent memory context
totalsize	bigint	Total size of the shared memory (unit: byte)
freesize	bigint	Remaining size of the shared memory (unit: byte)
usedsize	bigint	Used size of the shared memory (unit: byte)

12.3.12.94 GS_SQL_COUNT

GS_SQL_COUNT displays statistics about five types of running statements (SELECT, INSERT, UPDATE, DELETE, and MERGE INTO) on the current node of the database.

- When a common user queries the GS_SQL_COUNT view, statistics about the current node of the user are displayed. When an administrator queries the GS_SQL_COUNT view, statistics about the current node of all users are displayed.
- When the database or the node is restarted, the statistics are cleared and the counting restarts.
- The system counts when a node receives a query, including a query inside the database.

Table 12-318 GS_SQL_COUNT columns

Name	Type	Description
node_name	name	Node name.
user_name	name	Username.

Name	Type	Description
select_count	bigint	Statistical result of the SELECT statement.
update_count	bigint	Statistical result of the UPDATE statement.
insert_count	bigint	Statistical result of the INSERT statement.
delete_count	bigint	Statistical result of the DELETE statement.
mergeinto_count	bigint	Statistical result of the MERGE INTO statement.
ddl_count	bigint	Number of DDL statements.
dml_count	bigint	Number of DML statements.
dcl_count	bigint	Number of DML statements.
total_select_elapse	bigint	Total response time of SELECT statements (unit: μs).
avg_select_elapse	bigint	Average response time of SELECT statements (unit: μs).
max_select_elapse	bigint	Maximum response time of SELECT statements (unit: μs).
min_select_elapse	bigint	Minimum response time of SELECT statements (unit: μs).
total_update_elapse	bigint	Total response time of UPDATE statements (unit: μs).
avg_update_elapse	bigint	Average response time of UPDATE statements (unit: μs).
max_update_elapse	bigint	Maximum response time of UPDATE statements (unit: μs).
min_update_elapse	bigint	Minimum response time of UPDATE statements (unit: μs).
total_insert_elapse	bigint	Total response time of INSERT statements (unit: μs).
avg_insert_elapse	bigint	Average response time of INSERT statements (unit: μs).
max_insert_elapse	bigint	Maximum response time of INSERT statements (unit: μs).
min_insert_elapse	bigint	Minimum response time of INSERT statements (unit: μs).

Name	Type	Description
total_delete_elapse	bigint	Total response time of DELETE statements (unit: μs).
avg_delete_elapse	bigint	Average response time of DELETE statements (unit: μs).
max_delete_elapse	bigint	Maximum response time of DELETE statements (unit: μs).
min_delete_elapse	bigint	Minimum response time of DELETE statements (unit: μs).

12.3.12.95 GS_THREAD_MEMORY_CONTEXT

GS_THREAD_MEMORY_CONTEXT displays statistics about memory usage of all threads based on MemoryContext nodes. This view is equivalent to the GS_SESSION_MEMORY_DETAIL view when **enable_thread_pool** is set to **off**. If the GUC parameter **enable_memory_limit** is set to **off**, this view is unavailable.

The memory context **TempSmallContextGroup** collects information about all memory contexts whose value in the **totalsize** column is less than 8192 bytes in the current thread, and the number of the collected memory contexts is recorded in the **usedsize** column. Therefore, the **totalsize** and **freesize** columns for **TempSmallContextGroup** in the view display the corresponding information about all the memory contexts whose value in the **totalsize** column is less than 8192 bytes in the current thread, and the **usedsize** column displays the number of these memory contexts.

Table 12-319 GS_THREAD_MEMORY_CONTEXT columns

Name	Type	Description
threadid	text	Thread start time + thread ID (string: *timestamp.sessionid*).
tid	bigint	Thread ID.
thrdtype	text	Thread type.
contextname	text	Name of the memory context.
level	smallint	Hierarchy of the memory context.
parent	text	Name of the parent memory context.
totalsize	bigint	Total size of the memory context, in bytes.
freesize	bigint	Total size of released memory in the current memory context, in bytes.

Name	Type	Description
usedsize	bigint	Size of used memory in the memory context, in bytes. For **TempSmallContextGroup**, this parameter specifies the number of collected memory contexts.

12.3.12.96 GS_TOTAL_MEMORY_DETAIL

GS_TOTAL_MEMORY_DETAIL displays the memory usage of the current database node, in MB. If the GUC parameter **enable_memory_limit** is set to **off**, this view is unavailable.

Table 12-320 GS_TOTAL_MEMORY_DETAIL columns

Name	Type	Description
nodename	text	Node name.

Name	Type	Description
memorytype	text	Memory type. The value must be one of the following: • **max_process_memory**: memory occupied by the GaussDB instance • **process_used_memory**: memory occupied by the GaussDB process • **max_dynamic_memory**: maximum dynamic memory • **dynamic_used_memory**: used dynamic memory • **dynamic_peak_memory**: dynamic peak memory • **dynamic_used_shrctx**: maximum dynamic shared memory context • **dynamic_peak_shrctx**: dynamic peak value of the shared memory context • **max_backend_memory**: maximum memory that can be used when the HA port is used to execute services. • **backend_used_memory**: memory that has been used when the HA port is used to execute services. • **max_shared_memory**: maximum shared memory • **shared_used_memory**: used shared memory • **max_sctpcomm_memory**: maximum memory allowed for the communications library. • **sctpcomm_used_memory**: memory used by the communications library • **sctpcomm_peak_memory**: memory peak of the communications library • **other_used_memory**: other used memory • **llvm_used_memory**: memory occupied by the expression IR that is generated by Codegen and is not released
memorymbytes	integer	Size of allocated memory-typed memory.

12.3.12.97 GS_TOTAL_NODEGROUP_MEMORY_DETAIL

GS_TOTAL_NODEGROUP_MEMORY_DETAIL returns the memory usage (in MB) of the current logical instance of the database. If the GUC parameter **enable_memory_limit** is set to **off**, this view is unavailable.

Table 12-321 GS_TOTAL_NODEGROUP_MEMORY_DETAIL columns

Name	Type	Description
ngname	text	Logical instance name.
memorytype	text	Memory type. The value must be one of the following: • **ng_total_memory**: total memory of the logical instance • **ng_used_memory**: memory usage of the logical instance • **ng_estimate_memory**: estimated memory usage of the logical instance • **ng_foreignrp_memsize**: total memory of the external resource pool of the logical instance • **ng_foreignrp_usedsize**: memory usage of the external resource pool of the logical instance • **ng_foreignrp_peaksize**: peak memory usage of the external resource pool of the logical instance • **ng_foreignrp_mempct**: percentage of the external resource pool of the logical instance to the total memory of the logical instance • **ng_foreignrp_estmsize**: estimated memory usage of the external resource pool of the logical instance
memorymbytes	integer	Size of allocated memory-typed memory.

12.3.12.98 GS_WORKLOAD_RULE_STAT

GS_WORKLOAD_RULE_STAT displays information about SQL concurrency control rules. Only the sysadmin user can access the system view.

Table 12-322 GS_WORKLOAD_RULE_STAT columns

Name	Type	Description
rule_id	bigint	Concurrency control rule ID.
rule_name	name	Name of a concurrency control rule, which is used to search for the concurrency control rule.
databases	name[]	List of databases on which the concurrency control rules take effect. If the value is **NULL**, the concurrency control rules take effect for all databases.

Name	Type	Description
rule_type	text	Concurrency control rule type. Currently, only **"sqlid"**, **"select"**, **"insert"**, **"update"**, **"delete"**, **"merge"**, and **"resource"** are supported. Other values are invalid.
start_time	timestamp with time zone	Start time of the concurrency control rules. The value **NULL** indicates that the rules take effect from now on.
end_time	timestamp with time zone	End time of the concurrency control rules. The value **NULL** indicates that the rules are always effective.
max_workload	bigint	Maximum number of concurrent rule settings.
option_val	text[]	Parameter values of a concurrency control rule, including SQL ID, keyword list, and resource restriction. For details, see **gs_add_workload_rule(rule_type, rule_name, databases, start_time, end_time, max_workload, option_val)**.
is_valid	Boolean	Determines whether the concurrency control rules take effect. If the concurrency control rules time out, the value is set to **false**.
validate_count	bigint	Number of SQL statements intercepted by the concurrency control rule.
node_names	text[]	List of nodes on which the concurrency control rules take effect. This parameter is reserved and does not take effect currently.
user_names	text[]	List of users for which the concurrency control rules take effect. This parameter is reserved and does not take effect currently.

12.3.12.99 GV_INSTANCE

GV_INSTANCE displays information about the current database instance. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the PG_CATALOG and SYS schemas.

Table 12-323 GV_INSTANCE columns

Name	Type	Description
inst_id	oid	OID of the current database.
instance_number	oid	OID of the current database.
instance_name	character varying(16)	Name of the current database.
host_name	character varying(64)	Not supported. The value is **NULL**.
version	character varying(17)	Not supported. The value is **NULL**.
version_legacy	character varying(17)	Not supported. The value is **NULL**.
version_full	character varying(17)	Not supported. The value is **NULL**.
startup_time	timestamp(0) without time zone	Not supported. The value is **NULL**.
status	character varying(12)	Not supported. The value is **NULL**.
parallel	character varying(3)	Not supported. The value is **NULL**.
thread#	numeric	Not supported. The value is **NULL**.
archiver	character varying(7)	Not supported. The value is **NULL**.
log_switch_wait	character varying(15)	Not supported. The value is **NULL**.
logins	character varying(10)	Not supported. The value is **NULL**.
shutdown_pending	character varying(3)	Not supported. The value is **NULL**.
database_status	character varying(17)	Not supported. The value is **NULL**.
instance_role	character varying(18)	Not supported. The value is **NULL**.
active_state	character varying(9)	Not supported. The value is **NULL**.
blocked	character varying(3)	Not supported. The value is **NULL**.

Name	Type	Description
con_id	numeric	Not supported. The value is **NULL**.
instance_mode	character varying(11)	Not supported. The value is **NULL**.
edition	character varying(7)	Not supported. The value is **NULL**.
family	character varying(80)	Not supported. The value is **NULL**.
database_type	character varying(15)	Not supported. The value is **NULL**.

12.3.12.100 GV_SESSION

GV_SESSION displays information about all current sessions. Only administrators can access this view. Common users can access the view only after being authorized. This view exists in both the PG_CATALOG and SYS schemas. When the thread pool is enabled (**enable_thread_pool** is set to **on**), all session information is displayed. When the thread pool is disabled (**enable_thread_pool** is set to **off**), the sessions connected by users are not displayed.

Table 12-324 GV_SESSION columns

Name	Type	Description
inst_id	numeric	Not supported. The value is **NULL**.
saddr	raw	Not supported. The value is **NULL**.
sid	bigint	Session ID.
serial#	integer	Sequence number of the active background thread.
audsid	numeric	Not supported. The value is **NULL**.
paddr	raw	Not supported. The value is **NULL**.
schema#	numeric	Not supported. The value is **NULL**.
schemaname	name	Name of the user logged in to the backend.
user#	oid	OID of the user who has logged in to the backend thread. The OID is **0** if the backend thread is a global auxiliary thread.
username	name	Name of the user logged in to the backend process. **username** is null if the backend thread is a global auxiliary thread.

Name	Type	Description
command	numeric	Not supported. The value is **NULL**.
ownerid	numeric	Not supported. The value is **NULL**.
taddr	character varying(16)	Not supported. The value is **NULL**.
lockwait	character varying(16)	Not supported. The value is **NULL**.
machine	text	Host name of the connected client, as reported by a reverse DNS lookup of **client_addr**. This column will be non-null only for IP connections and only when **log_hostname** is enabled.
sql_id	bigint	SQL OID.
client_info	text	Client information
event	text	Queuing status of a statement. The value can be: • **waiting in queue**: The statement is in the queue. • **Empty**: The statement is running.
sql_exec_start	timestamp with time zone	Start time of SQL statement execution.
program	text	Name of the application connected to the backend.
status	text	Overall status of this backend. The value can be: • **active**: The backend is executing a query. • **idle**: The backend is waiting for a new client command. • **idle in transaction**: The backend is in a transaction, but there is no statement being executed in the transaction. • **idle in transaction (aborted)**: The backend is in a transaction, but there are statements failed in the transaction. • **fastpath function call**: The backend is executing a fast-path function. • **disabled**: This state is reported if **track_activities** is disabled in this backend.
server	character varying(9)	Not supported. The value is **NULL**.
pdml_status	character varying(8)	Specifies whether to enable a DML parallel execution in the current session.

Name	Type	Description
port	numeric	Port number of the current session
process	character varying(24)	Process ID of the current session
logon_time	timestamp(0) without time zone	Login time of the current session
last_call_et	integer	Duration when the status of the current session changes last time
osuser	text	OS username of the server.
terminal	character varying(30)	Not supported. The value is **NULL**.
type	character varying(10)	Not supported. The value is **NULL**.
sql_address	raw	Not supported. The value is **NULL**.
sql_hash_value	numeric	Not supported. The value is **NULL**.
sql_child_number	numeric	Not supported. The value is **NULL**.
sql_exec_id	numeric	Not supported. The value is **NULL**.
prev_sql_addr	raw	Not supported. The value is **NULL**.
prev_hash_value	numeric	Not supported. The value is **NULL**.
prev_sql_id	character varying(13)	Not supported. The value is **NULL**.
prev_child_number	numeric	Not supported. The value is **NULL**.
prev_exec_start	timestamp(0) without time zone	Not supported. The value is **NULL**.
prev_exec_id	numeric	Not supported. The value is **NULL**.
plsql_entry_object_id	numeric	Not supported. The value is **NULL**.
plsql_entry_subprogram_id	numeric	Not supported. The value is **NULL**.

Name	Type	Description
plsql_object_id	numeric	Not supported. The value is **NULL**.
plsql_subprogram_id	numeric	Not supported. The value is **NULL**.
module	text	Name of the running module, which is set by calling DBE_APPLICATION_INFO.SET_MODULE.
module_hash	numeric	Not supported. The value is **NULL**.
action	text	Name of the current operation in the current module, which is set by calling DBE_APPLICATION_INFO.SET_MODULE or DBE_APPLICATION_INFO.SET_ACTION.
action_hash	numeric	Not supported. The value is **NULL**.
fixed_table_sequence	numeric	Not supported. The value is **NULL**.
row_wait_obj#	numeric	Not supported. The value is **NULL**.
row_wait_file#	numeric	Not supported. The value is **NULL**.
row_wait_block#	numeric	Not supported. The value is **NULL**.
row_wait_row#	numeric	Not supported. The value is **NULL**.
top_level_call#	numeric	Not supported. The value is **NULL**.
pdml_enabled	character varying(3)	Not supported. The value is **NULL**.
failover_type	character varying(13)	Not supported. The value is **NULL**.
failover_method	character varying(10)	Not supported. The value is **NULL**.
failed_over	character varying(3)	Not supported. The value is **NULL**.
resource_consumer_group	character varying(32)	Not supported. The value is **NULL**.
pddl_status	character varying(8)	Not supported. The value is **NULL**.

Name	Type	Description
pq_status	character varying(8)	Not supported. The value is **NULL**.
current_queue_duration	numeric	Not supported. The value is **NULL**.
client_identifier	character varying(64)	Not supported. The value is **NULL**.
blocking_session_status	character varying(11)	Not supported. The value is **NULL**.
blocking_instance	numeric	Not supported. The value is **NULL**.
blocking_session	numeric	Not supported. The value is **NULL**.
final_blocking_session_status	character varying(11)	Not supported. The value is **NULL**.
final_blocking_instance	numeric	Not supported. The value is **NULL**.
final_blocking_session	numeric	Not supported. The value is **NULL**.
seq#	numeric	Not supported. The value is **NULL**.
event#	numeric	Not supported. The value is **NULL**.
p1text	character varying(64)	Not supported. The value is **NULL**.
p1	numeric	Not supported. The value is **NULL**.
p1raw	raw	Not supported. The value is **NULL**.
p2text	character varying(64)	Not supported. The value is **NULL**.
p2	numeric	Not supported. The value is **NULL**.
p2raw	raw	Not supported. The value is **NULL**.
p3text	character varying(64)	Not supported. The value is **NULL**.
p3	numeric	Not supported. The value is **NULL**.
p3raw	raw	Not supported. The value is **NULL**.
wait_class_id	numeric	Not supported. The value is **NULL**.
wait_class#	numeric	Not supported. The value is **NULL**.

Name	Type	Description
wait_class	character varying(64)	Not supported. The value is **NULL**.
wait_time	numeric	Not supported. The value is **NULL**.
seconds_in_wait	numeric	Not supported. The value is **NULL**.
state	character varying(19)	Not supported. The value is **NULL**.
wait_time_micro	numeric	Not supported. The value is **NULL**.
time_remaining_micro	numeric	Not supported. The value is **NULL**.
time_since_last_wait_micro	numeric	Not supported. The value is **NULL**.
service_name	character varying(64)	Not supported. The value is **NULL**.
sql_trace	character varying(8)	Not supported. The value is **NULL**.
sql_trace_waits	character varying(5)	Not supported. The value is **NULL**.
sql_trace_binds	character varying(5)	Not supported. The value is **NULL**.
sql_trace_plan_stats	character varying(10)	Not supported. The value is **NULL**.
session_edition_id	numeric	Not supported. The value is **NULL**.
creator_addr	raw	Not supported. The value is **NULL**.
creator_serial#	numeric	Not supported. The value is **NULL**.
ecid	character varying(64)	Not supported. The value is **NULL**.
sql_translation_profile_id	numeric	Not supported. The value is **NULL**.
pga_tunable_mem	numeric	Not supported. The value is **NULL**.
shard_ddl_status	character varying(8)	Not supported. The value is **NULL**.

Name	Type	Description
con_id	numeric	Not supported. The value is **NULL**.
external_name	character varying(1024)	Not supported. The value is **NULL**.
plsql_debugger_connected	character varying(5)	Not supported. The value is **NULL**.

12.3.12.101 MPP_TABLES

The following information is displayed in the MPP_TABLES view.

Table 12-325 MPP_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that contains a table.
tablename	name	Table name.
tableowner	name	Table owner.
tablespace	name	Tablespace that contains the table.
pgroup	name	Name of a node cluster.
nodeoids	oidvector_extend	List of distributed table node OIDs.

12.3.12.102 MY_AUDIT_TRAIL

MY_AUDIT_TRAIL displays the standard audit trail entries related to the current user. The GaussDB audit information is mainly obtained through the pg_query_audit() function. This view exists in both the PG_CATALOG and SYS schemas. Only users with the AUDITADMIN attribute and the SELECT permission on MY_AUDIT_TRAIL can view audit information. If separation-of-duty is disabled, users with the SYSADMIN attribute can also view audit information. The **action_name** field of GaussDB is different from that of the A database in terms of the audit actions. The type of the **transactionid** field is the same as that in the A database. In GaussDB, the **sql_text** field is the parsed SQL statement, which is not completely the same as the executed SQL statement.

Table 12-326 MY_AUDIT_TRAIL columns

Name	Type	Description
os_username	character varying(255)	Not supported. The value is **NULL**.
username	character varying(128)	Name of the user whose operation is audited, not the user ID.
userhost	character varying(128)	Not supported. The value is **NULL**.
terminal	character varying(255)	Not supported. The value is **NULL**.
timestamp	timestamp(0) without time zone	Date and time when an audit trail entry is created in the local database session time zone (user login date and time of the entry created by the audit session).
owner	character varying(128)	Creator of the object affected by the operation.
obj_name	character varying(128)	Name of the object affected by the operation.
action	numeric	Not supported. The value is **NULL**.
action_name	character varying(28)	Action type corresponding to the code in the **action** column.
new_owner	character varying(128)	Not supported. The value is **NULL**.
new_name	character varying(128)	Not supported. The value is **NULL**.
obj_privilege	character varying(32)	Not supported. The value is **NULL**.
sys_privilege	character varying(40)	Not supported. The value is **NULL**.
admin_option	character varying(1)	Not supported. The value is **NULL**.
grantee	character varying(128)	Not supported. The value is **NULL**.
audit_option	character varying(40)	Not supported. The value is **NULL**.
ses_actions	character varying(19)	Not supported. The value is **NULL**.

Name	Type	Description
logoff_time	timestamp(0) without time zone	Not supported. The value is **NULL**.
logoff_lread	numeric	Not supported. The value is **NULL**.
logoff_pread	numeric	Not supported. The value is **NULL**.
logoff_lwrite	numeric	Not supported. The value is **NULL**.
logoff_dlock	character varying(40)	Not supported. The value is **NULL**.
comment_text	character varying(4000)	Not supported. The value is **NULL**.
sessionid	numeric	Not supported. The value is **NULL**.
entryid	numeric	Not supported. The value is **NULL**.
statementid	numeric	Not supported. The value is **NULL**.
returncode	numeric	Not supported. The value is **NULL**.
priv_used	character varying(40)	Not supported. The value is **NULL**.
client_id	character varying(128)	Not supported. The value is **NULL**.
econtext_id	character varying(64)	Not supported. The value is **NULL**.
session_cpu	numeric	Not supported. The value is **NULL**.
extended_timestamp	timestamp(6) with time zone	Time zone of the timestamp when an audit trail entry is created (user login date and time of the entry created by the audit session in UTC).
proxy_sessionid	numeric	Not supported. The value is **NULL**.
global_uid	character varying(32)	Not supported. The value is **NULL**.
instance_number	numeric	Not supported. The value is **NULL**.
os_process	character varying(16)	Not supported. The value is **NULL**.
transactionid	text	Identifier of the transaction that accesses or modifies an object.
scn	numeric	Not supported. The value is **NULL**.
sql_bind	nvarchar2(2000)	Not supported. The value is **NULL**.

Name	Type	Description
sql_text	nvarchar2	SQL text of the query.
obj_edition_name	character varying(128)	Not supported. The value is **NULL**.
dbid	numeric	Not supported. The value is **NULL**.
rls_info	clob	Not supported. The value is **NULL**.
current_user	character varying(128)	Not supported. The value is **NULL**.

12.3.12.103 MY_COL_COMMENTS

MY_COL_COMMENTS displays column comments of the table accessible to the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-327 MY_COL_COMMENTS columns

Name	Type	Description
owner	character varying(128)	Table owner.
table_name	character varying(128)	Table name.
column_name	character varying(128)	Column name.
comments	text	Comments.
origin_con_id	numeric	Not supported. The value is **0**.
schema	character varying(64)	Name of the namespace to which the column belongs.

12.3.12.104 MY_COLL_TYPES

MY_COLL_TYPES displays information about types of collections created by the current user. By default, it is accessible to all users. This view exists in the PG_CATALOG and SYS schemas.

Table 12-328 MY_COLL_TYPES columns

Name	Type	Description
owner	character varying(128)	Owner of a collection.
type_name	character varying(128)	Name of a collection.

Name	Type	Description
coll_type	character varying(128)	Description of a collection.
upper_bound	numeric	Not supported. The value is **NULL**.
elem_type_mod	character varying(7)	Type modifier of an element.
elem_type_owner	character varying(128)	Owner of the element type on which the collection is based. This parameter is mainly used for user-defined types.
elem_type_name	character varying(128)	Name of the data type or user-defined type on which the collection is based.
length	numeric	Not supported. The value is **NULL**.
precision	numeric	Not supported. The value is **NULL**.
scale	numeric	Not supported. The value is **NULL**.
character_set_name	character varying(44)	Not supported. The value is **NULL**.
elem_storage	character varying(7)	Not supported. The value is **NULL**.
nulls_stored	character varying(3)	Not supported. The value is **NULL**.
char_used	character varying(1)	Not supported. The value is **NULL**.

12.3.12.105 MY_CONS_COLUMNS

MY_CONS_COLUMNS displays information about primary key constraint columns in tables accessible to the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-329 MY_CONS_COLUMNS columns

Name	Type	Description
table_name	character varying(64)	Name of a constraint-related table.
column_name	character varying(64)	Name of a constraint-related column.
constraint_name	character varying(64)	Constraint name.
owner	character varying(64)	Constraint creator.
position	smallint	Position of a column in a table.

12.3.12.106 MY_CONSTRAINTS

MY_CONSTRAINTS displays table constraint information accessible to the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-330 MY_CONSTRAINTS columns

Name	Type	Description
constraint_name	character varying(64)	Constraint name.
constraint_type	text	Constraint type. • **c**: check constraint • **f**: foreign key constraint • **p**: primary key constraint • **u**: unique constraint
table_name	character varying(64)	Name of a constraint-related table.
index_owner	character varying(64)	Owner of a constraint-related index (only for the unique constraint and primary key constraint).
index_name	character varying(64)	Name of a constraint-related index (only for the unique constraint and primary key constraint).
owner	character varying(64)	Constraint creator.

12.3.12.107 MY_DEPENDENCIES

MY_DEPENDENCIES displays the dependencies between objects that are accessible to the current user. This view exists in the PG_CATALOG and SYS schemas and all users can access this view.

Table 12-331 MY_DEPENDENCIES columns

Name	Type	Description
name	name	Object name.
type	character varying(18)	Object type.
referenced_owner	name	Owner of the referenced object.
referenced_name	name	Name of the referenced object.
referenced_type	character varying(18)	Type of the referenced object.
referenced_link_name	character varying(128)	Name of the link to the parent object (if remote).
dependency_type	character varying(4)	Specifies whether the dependency is an REF dependency.

12.3.12.108 MY_ERRORS

MY_ERRORS displays the latest compilation error information about user-owned storage objects. This view can be accessed by all users. Only the information about the current user can be viewed. This view exists in the PG_CATALOG and SYS schemas.

Table 12-332 MY_ERRORS columns

Name	Type	Description
name	character varying(128)	Object name.
type	character varying(12)	Object type. • **PROCEDURE** • **FUNCTION** • **PACKAGE** • **PACKAGE BODY**

Name	Type	Description
sequence	numeric	Serial number.
line	numeric	Row where an error occurs.
position	numeric	Position in the row where an error occurs.
text	character varying(4000)	Error text.
attribute	character varying(9)	Attribute tag: ERROR.
message_number	numeric	Not supported. The value is **NULL**.

12.3.12.109 MY_IND_COLUMNS

MY_IND_COLUMNS displays column information about all indexes accessible to the current user. It is accessible to all users. This view exists in the PG_CATALOG and SYS schemas.

Table 12-333 MY_IND_COLUMNS columns

Name	Type	Description
index_owner	character varying(64)	Index owner.
index_name	character varying(64)	Index name.
table_owner	character varying(64)	Table owner.
table_name	character varying(64)	Table name.
column_name	name	Column name.
column_position	smallint	Position of a column in an index.
column_length	numeric	Length of the column. For the variable-length type, the value of this field is **NULL**.
char_length	numeric	Maximum length of a column, in bytes.
descend	character varying(4)	Specifies whether columns are sorted in descending (**DESC**) or ascending (**ASC**) order.
collated_column_id	numeric	Not supported. The value is **NULL**.

12.3.12.110 MY_IND_EXPRESSIONS

MY_IND_EXPRESSIONS displays information about function-based expression indexes accessible to the current user. It is accessible to all users. This view exists in the PG_CATALOG and SYS schemas.

Table 12-334 MY_IND_EXPRESSIONS columns

Name	Type	Description
table_owner	character varying(64)	Table owner.
table_name	character varying(64)	Table name.
index_owner	character varying(64)	Index owner.
index_name	character varying(64)	Index name.
column_expression	text	Function-based index expression of a specified column.
column_position	smallint	Position of a column in an index.

12.3.12.111 MY_INDEXES

MY_INDEXES displays index information about the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-335 MY_INDEXES columns

Name	Type	Description
owner	character varying(64)	Index owner.
index_name	character varying(64)	Index name.
table_name	character varying(64)	Name of a table corresponding to an index.
uniqueness	text	Specifies whether an index is unique.
partitioned	character(3)	Specifies whether an index has the property of partitioned tables.
generated	character varying(1)	Specifies whether the name of an index is generated by the system.
index_type	character varying(27)	Index type.
table_owner	character varying(128)	Owner of an index object.

Name	Type	Description
table_type	character(11)	Type of an index object.
tablespace_name	character varying(30)	Name of the tablespace that contains the index.
status	character varying(8)	Status of a non-partitioned index.
compression	character varying(13)	Not supported. The value is **NULL**.
prefix_length	numeric	Not supported. The value is **NULL**.
ini_trans	numeric	Not supported. The value is **NULL**.
max_trans	numeric	Not supported. The value is **NULL**.
initial_extent	numeric	Not supported. The value is **NULL**.
next_extent	numeric	Not supported. The value is **NULL**.
min_extents	numeric	Not supported. The value is **NULL**.
max_extents	numeric	Not supported. The value is **NULL**.
pct_increase	numeric	Not supported. The value is **NULL**.
pct_threshold	numeric	Not supported. The value is **NULL**.
include_column	numeric	Not supported. The value is **NULL**.
freelists	numeric	Not supported. The value is **NULL**.
freelist_groups	numeric	Not supported. The value is **NULL**.
pct_free	numeric	Not supported. The value is **NULL**.
logging	character varying(3)	Not supported. The value is **NULL**.
blevel	numeric	Not supported. The value is **NULL**.

Name	Type	Description
leaf_blocks	numeric	Not supported. The value is **NULL**.
distinct_keys	numeric	Not supported. The value is **NULL**.
avg_leaf_blocks_per_key	numeric	Not supported. The value is **NULL**.
avg_data_blocks_per_key	numeric	Not supported. The value is **NULL**.
clustering_factor	numeric	Not supported. The value is **NULL**.
num_rows	numeric	Not supported. The value is **NULL**.
sample_size	numeric	Not supported. The value is **NULL**.
last_analyzed	timestamp(0) without time zone	Not supported. The value is **NULL**.
degree	character varying(40)	Not supported. The value is **NULL**.
instances	character varying(40)	Not supported. The value is **NULL**.
temporary	character varying(1)	Not supported. The value is **NULL**.
secondary	character varying(1)	Not supported. The value is **NULL**.
buffer_pool	character varying(7)	Not supported. The value is **NULL**.
flash_cache	character varying(7)	Not supported. The value is **NULL**.
cell_flash_cache	character varying(7)	Not supported. The value is **NULL**.
user_stats	character varying(3)	Not supported. The value is **NULL**.
duration	character varying(15)	Not supported. The value is **NULL**.
pct_direct_access	numeric	Not supported. The value is **NULL**.
ityp_owner	character varying(128)	Not supported. The value is **NULL**.

Name	Type	Description
ityp_name	character varying(128)	Not supported. The value is **NULL**.
parameters	character varying(1000)	Not supported. The value is **NULL**.
global_stats	character varying(3)	Not supported. The value is **NULL**.
domidx_status	character varying(12)	Not supported. The value is **NULL**.
domidx_opstatus	character varying(6)	Not supported. The value is **NULL**.
funcidx_status	character varying(8)	Not supported. The value is **NULL**.
join_index	character varying(3)	Not supported. The value is **NULL**.
iot_redundant_pkey_elim	character varying(3)	Not supported. The value is **NULL**.
dropped	character varying(3)	Not supported. The value is **NULL**.
visibility	character varying(9)	Specifies whether the index is visible to the optimizer.
domidx_management	character varying(14)	Not supported. The value is **NULL**.
segment_created	character varying(3)	Not supported. The value is **NULL**.
orphaned_entries	character varying(3)	Not supported. The value is **NULL**.
indexing	character varying(7)	Not supported. The value is **NULL**.
auto	character varying(3)	Not supported. The value is **NULL**.

12.3.12.112 MY_JOBS

MY_JOBS displays details about the scheduled tasks owned by the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-336 MY_JOBS columns

Name	Type	Description
job	bigint	Job ID.
log_user	name	Username of a job creator.
priv_user	name	Username of a job executor.
dbname	name	Name of the database where the job is created.
schema_user	name	Default schema name of a scheduler job.
start_date	timestamp without time zone	Job start time.
start_suc	text	Start time of a successful job execution.
last_date	timestamp without time zone	Start time of the last job execution.
last_suc	text	Start time of the last successful job execution.
last_sec	text	Start time of the last successful job execution. Compatibility is supported.
this_date	timestamp without time zone	Start time of an ongoing job execution.
this_suc	text	Start time of an ongoing successful job execution.
this_sec	text	Start time of an ongoing successful job execution. Compatibility is supported.
next_date	timestamp without time zone	Schedule time of the next job execution.
next_suc	text	Schedule time of the next successful job execution.
next_sec	text	Schedule time of the next successful job execution. Compatibility is supported.
total_time	numeric	Latest execution duration of a task.
broken	text	The value is **y** if the job state is broken and **n** if otherwise.

Name	Type	Description
status	"char"	Status of the current job. The value can be **r**, **s**, **f**, or **d**. The default value is **r**. • **r**: running • **s**: successful execution • **f**: failed execution • **d**: canceling an execution
interval	text	Time expression used to calculate the next time the job will be executed. If this parameter is set to **null**, the job will be executed once only.
failures	smallint	Number of times the job has started and failed. If a job fails to be executed for 16 consecutive times, no more attempt will be made on it.
what	text	Executable job.
nls_env	character varying(4000)	Not supported. The value is **NULL**.
misc_env	raw	Not supported. The value is **NULL**.
instance	numeric	Not supported. The value is **NULL**.

12.3.12.113 MY_OBJECTS

MY_OBJECTS displays database objects accessible to the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-337 MY_OBJECTS columns

Name	Type	Description
object_name	name	Object name.
object_id	oid	Object OID.
object_type	name	Object type (**TABLE**, **INDEX**, **SEQUENCE**, or **VIEW**).
namespace	oid	Namespace that an object belongs to.
temporary	character(1)	Specifies whether an object is a temporary object.

Name	Type	Description
status	character varying(7)	Object status.
subobject_name	name	Subobject name of an object.
generated	character(1)	Specifies whether an object name is generated by the system.
created	timestamp with time zone	Creation time of an object.
last_ddl_time	timestamp with time zone	Last modification time of an object.
default_collation	character varying(100)	Default collation of objects.
data_object_id	numeric	Not supported. The value is **NULL**.
timestamp	character varying(19)	Not supported. The value is **NULL**.
secondary	character varying(1)	Not supported. The value is **NULL**.
edition_name	character varying(128)	Not supported. The value is **NULL**.
sharing	character varying(18)	Not supported. The value is **NULL**.
editionable	character varying(1)	Not supported. The value is **NULL**.
oracle_maintained	character varying(1)	Not supported. The value is **NULL**.
application	character varying(1)	Not supported. The value is **NULL**.
duplicated	character varying(1)	Not supported. The value is **NULL**.
sharded	character varying(1)	Not supported. The value is **NULL**.
created_appid	numeric	Not supported. The value is **NULL**.
modified_appid	numeric	Not supported. The value is **NULL**.

Name	Type	Description
created_vsnid	numeric	Not supported. The value is **NULL**.
modified_vsnid	numeric	Not supported. The value is **NULL**.

NOTICE

For details about the value ranges of **created** and **last_ddl_time**, see **PG_OBJECT**.

12.3.12.114 MY_PROCEDURES

MY_PROCEDURES displays information about stored procedures, functions, or triggers owned by the current user. This view exists in the PG_CATALOG and SYS schemas. This view can be accessed by all users. Only the information about the current user can be viewed.

Table 12-338 MY_PROCEDURES columns

Name	Type	Description
owner	character varying(64)	Owner of a stored procedure, function, trigger, or package.
object_name	character varying(64)	Name of a stored procedure, function, or trigger. This column will be a package name if the object is a function in a package or a stored procedure.
procedure_name	character varying(128)	This column will be the name of a function or stored procedure in a package if **object_name** is a package name. Otherwise, this column is empty.
object_id	oid	OID of a stored procedure, function, trigger, or package.
subprogram_id	numeric	Location of a function or stored procedure in a package.
overload	character varying(40)	*M*th overloaded function.
object_type	character varying(13)	Object class.

Name	Type	Description
aggregate	character varying(3)	Specifies whether the function is an aggregate function: • **YES** • **NO**
pipelined	character varying(3)	Not supported. The value is **NO**.
impltypeowner	character varying(128)	Owner of an implementation type.
impltypename	character varying(128)	Name of an implementation type.
parallel	character varying(3)	Not supported. The value is **NO**.
interface	character varying(3)	Not supported. The value is **NO**.
deterministic	character varying(3)	Not supported. The value is **NO**.
authid	character varying(12)	Specifies whether to use the creator permission or caller permission: • **DEFINER**: The creator permission is used. • **CURRENT_USER**: The caller permission is used. This column conflicts with reserved keywords. Therefore, add the view name when calling this column.
result_cache	character varying(3)	Not supported. The value is **NULL**.
origin_con_id	character varying(256)	Not supported. The value is **0**.
polymorphic	character varying(5)	Not supported. The value is **NULL**.
argument_number	smallint	Number of input parameters in a stored procedure.

12.3.12.115 MY_RECYCLEBIN

MY_RECYCLEBIN displays information about the recycle bin owned by the current user. This view exists in the PG_CATALOG and SYS schemas and all users can access this view.

Table 12-339 MY_RECYCLEBIN columns

Name	Type	Description
object_name	character varying(128)	New name of an object.
original_name	character varying(128)	Original name of an object.
operation	character varying(18)	Operation performed on an object. The options are as follows: ● **DROP**: The object is deleted (the object is no longer required). ● **TRUNCATE**: The object is truncated.
type	character varying(25)	Object class.
ts_name	character varying(30)	Name of the tablespace to which the object belongs.
createtime	character varying(19)	Timestamp when an object is created.
droptime	character varying(19)	Timestamp when an object is deleted.
dropscn	numeric	System change number (SCN) of the transaction that moves an object to the recycle bin.
partition_name	character varying(128)	Name of a deleted partition.
can_undrop	character varying(3)	Determines whether to flash back an object.
can_purge	character varying(3)	Determines whether to purge the object.
related	numeric	ID of the parent object.
base_object	oid	ID of a base object.
purge_object	oid	ID of the purged object.
space	numeric	Number of blocks used by an object.

12.3.12.116 MY_SCHEDULER_JOB_ARGS

MY_SCHEDULER_JOB_ARG displays the parameters related to the jobs owned by the current user. This view can be accessed by all users. Only the information about the current user can be viewed. This view exists in the PG_CATALOG and SYS schemas.

Table 12-340 MY_SCHEDULER_JOB_ARGS columns

Name	Type	Description
job_name	character varying(128)	Name of the job to which the parameter belongs.
argument_name	character varying(128)	Parameter name.
argument_position	numeric	Position of the parameter in the parameter list.
argument_type	character varying(257)	Data type of a parameter.
value	character varying(4000)	Parameter value.
anydata_value	character varying(4000)	Not supported. The value is **NULL**.
out_argument	character varying(5)	This column is reserved. The value is **NULL**.

12.3.12.117 MY_SCHEDULER_JOBS

MY_SCHEDULER_JOBS displays information about all DBE_SCHEDULER scheduled tasks owned by the current user in the database. By default, it is accessible to all users. This view exists in the PG_CATALOG and SYS schemas.

Table 12-341 MY_SCHEDULER_JOBS columns

Name	Type	Description
job_name	text	Name of a scheduler job.
job_subname	character varying(128)	Not supported. The value is **NULL**.
job_style	text	Action mode of a scheduler job.
job_creator	name	Creator of a scheduler job.
client_id	character varying(65)	Not supported. The value is **NULL**.

Name	Type	Description
global_uid	character varying(33)	Not supported. The value is **NULL**.
program_owner	character varying(4000)	Owner of a program referenced by a scheduler job.
program_name	text	Name of the program referenced by a scheduler job.
job_type	character varying(16)	Type of the inline program of a scheduler job.
job_action	text	Program content of a scheduler job.
number_of_arguments	text	Number of parameters of a scheduler job.
schedule_owner	character varying(4000)	Not supported. The value is **NULL**.
schedule_name	text	Name of the scheduler referenced by a scheduler job.
schedule_type	character varying(12)	Not supported. The value is **NULL**.
start_date	timestamp without time zone	Start time of a scheduler job.
repeat_interval	text	Period of a scheduler job.
event_queue_owner	character varying(128)	Not supported. The value is **NULL**.
event_queue_name	character varying(128)	Not supported. The value is **NULL**.
event_queue_agent	character varying(523)	Not supported. The value is **NULL**.
event_condition	character varying(4000)	Not supported. The value is **NULL**.
event_rule	character varying(261)	Not supported. The value is **NULL**.
file_watcher_owner	character varying(261)	Not supported. The value is **NULL**.
file_watcher_name	character varying(261)	Not supported. The value is **NULL**.

Name	Type	Description
end_date	timestamp without time zone	End time of a scheduler job.
job_class	text	Name of the scheduler job class to which a scheduler job belongs.
enabled	boolean	Status of a scheduler job.
auto_drop	text	Status of the automatic deletion function of a scheduler job.
restart_on_recovery	character varying(5)	Not supported. The value is **NULL**.
restart_on_failure	character varying(5)	Not supported. The value is **NULL**.
state	"char"	Status of a scheduler job.
job_priority	numeric	Not supported. The value is **NULL**.
run_count	numeric	Not supported. The value is **NULL**.
uptime_run_count	numeric	Not supported. The value is **NULL**.
max_runs	numeric	Not supported. The value is **NULL**.
failure_count	smallint	Number of scheduler job failures.
uptime_failure_count	numeric	Not supported. The value is **NULL**.
max_failures	numeric	Maximum number of failures allowed before the status of a scheduler job is marked as broken.
retry_count	numeric	Not supported. The value is **NULL**.
last_start_date	timestamp without time zone	Last time when a scheduler job was started.
last_run_duration	interval day to second(6)	Last execution duration of a scheduler job.
next_run_date	timestamp without time zone	Next execution time of a scheduler job.

Name	Type	Description
schedule_limit	interval day to second(0)	Not supported. The value is **NULL**.
max_run_duration	interval day to second(0)	Not supported. The value is **NULL**.
logging_level	character varying(11)	Not supported. The value is **NULL**.
store_output	character varying(5)	Specifies whether to store the output information of all scheduler jobs.
stop_on_window_close	character varying(5)	Not supported. The value is **NULL**.
instance_stickiness	character varying(5)	Not supported. The value is **NULL**.
raise_events	character varying(4000)	Not supported. The value is **NULL**.
system	character varying(5)	Not supported. The value is **NULL**.
job_weight	numeric	Not supported. The value is **NULL**.
nls_env	character varying(4000)	Not supported. The value is **NULL**.
source	character varying(128)	Not supported. The value is **NULL**.
number_of_destinations	numeric	Not supported. The value is **NULL**.
destination_owner	character varying(261)	Not supported. The value is **NULL**.
destination	text	Target name of a scheduler job.
credential_owner	character varying(128)	Not supported. The value is **NULL**.
credential_name	text	Certificate name of a scheduler job.
instance_id	oid	OID of the current database.
deferred_drop	character varying(5)	Not supported. The value is **NULL**.
allow_runs_in_restricted_mode	character varying(5)	Not supported. The value is **NULL**.

Name	Type	Description
comments	text	Comments of a scheduler job.
flags	numeric	Not supported. The value is **NULL**.
restartable	character varying(5)	Not supported. The value is **NULL**.
has_constraints	character varying(5)	Not supported. The value is **NULL**.
connect_credential_owner	character varying(128)	Not supported. The value is **NULL**.
connect_credential_name	character varying(128)	Not supported. The value is **NULL**.
fail_on_script_error	character varying(5)	Not supported. The value is **NULL**.

12.3.12.118 MY_SCHEDULER_PROGRAM_ARGS

MY_SCHEDULER_PROGRAM_ARG displays the parameters related to the programs owned by the current user. This view can be accessed by all users. Only the information about the current user can be viewed. This view exists in the PG_CATALOG and SYS schemas.

Table 12-342 MY_SCHEDULER_PROGRAM_ARGS columns

Name	Type	Description
program_name	character varying(128)	Name of the program to which the parameter belongs.
argument_name	character varying(128)	Parameter name.
argument_position	numeric	Position of the parameter in the parameter list.
argument_type	character varying(257)	Data type of a parameter.
metadata_attribute	character varying(19)	Not supported. Set it to **NULL**.
default_value	character varying(4000)	Default parameter value.
default_anydata_value	character varying(4000)	Not supported. Set it to **NULL**.

Name	Type	Description
out_argument	character varying(5)	Reserved column. Set it to **NULL**.

12.3.12.119 MY_SEQUENCES

MY_SEQUENCES displays the sequence information about the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-343 MY_SEQUENCES columns

Name	Type	Description
sequence_owner	name	Owner of a sequence.
sequence_name	name	Name of a sequence.
min_value	int16	Minimum value of a sequence.
max_value	int16	Maximum value of a sequence.
increment_by	int16	Value by which a sequence is incremented.
cycle_flag	character(1)	Whether a sequence is a cycle sequence. The value can be **Y** or **N**. • **Y**: It is a cycle sequence. • **N**: It is not a cycle sequence.
order_flag	character varying(1)	Specifies whether a sequence occurs in a request sequence. This parameter is not supported. Set it to **NULL**.
cache_size	int16	Size of the sequence disk cache.
last_number	int16	Value of the previous sequence.
scale_flag	character varying(1)	Specifies whether a sequence is a scalable sequence. This parameter is not supported. Set it to **NULL**.

Name	Type	Description
extend_flag	character varying(1)	Specifies whether the value generated by a scalable sequence exceeds the maximum or minimum value of the sequence. This parameter is not supported. Set it to **NULL**.
sharded_flag	character varying(1)	Specifies whether a sequence is a shard sequence. This parameter is not supported. Set it to **NULL**.
session_flag	character varying(1)	Specifies whether a sequence has a private session. This parameter is not supported. Set it to **NULL**.
keep_value	character varying(1)	Specifies whether to retain the sequence value during replay after a failure. This parameter is not supported. Set it to **NULL**.

12.3.12.120 MY_SOURCE

MY_SOURCE displays the definition information about stored procedures, functions, triggers, and packages accessible to the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-344 MY_SOURCE columns

Name	Type	Description
owner	name	Object owner.
name	name	Object name.
type	name	Object types: **function**, **package**, **package body**, **procedure**, and **trigger**
line	numeric	Number of the source line.
text	text	Text source of the storage object.
origin_con_id	character varying(256)	Not supported. The value is **0**.

12.3.12.121 MY_SYNONYMS

MY_SYNONYMS displays synonyms in the current schema. This view can be accessed by all users. Only the information about the current user can be viewed. This view exists in the PG_CATALOG and SYS schemas.

Table 12-345 MY_SYNONYMS columns

Name	Type	Description
schema_name	text	Name of the schema to which the synonym belongs.
synonym_name	text	Synonym name.
table_owner	text	Owner of the associated object. Although the column is called **table_owner**, the associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.
table_name	text	Name of the associated object. Although the column is called **table_name**, the associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.
table_schema_name	text	Schema name of the associated object. Although the column is called **table_schema_name**, the associated object is not necessarily a table. It can be any common database object, such as a view, stored procedure, or synonym.
db_link	character varying(128)	This column is reserved. The value is **NULL**.
origin_con_id	character varying(256)	Not supported. The value is **0**.

12.3.12.122 MY_TAB_COL_STATISTICS

MY_TAB_COL_STATISTICS displays column statistics and histogram information extracted from MY_TAB_COLUMNS. All users can access this view. This view exists in the PG_CATALOG and SYS schemas. The values of the **LOW_VALUE** and **HIGH_VALUE** columns in this view are different from those in database A due to different underlying table structures. When the value of **LOW_VALUE** is a high-

frequency value, the value of **LOW_VALUE** in GaussDB is the second minimum value. When **HIGH_VALUE** is a high-frequency value, **HIGH_VALUE** of GaussDB is the second maximum value. The value of the **HISTOGRAM** column is different from that of database A due to different statistical methods. GaussDB supports only two types of histograms: **frequency** and **equi-width**. The value of the **SCOPE** column in GaussDB is different from that in database A because GaussDB does not support global temporary table statistics. GaussDB supports only local temporary table statistics. The default value is **SHARED**.

Table 12-346 MY_TAB_COL_STATISTICS columns

Name	Type	Description
table_name	character varying(128)	Table name.
column_name	character varying(128)	Column name.
num_distinct	numeric	Number of different values in a column.
low_value	raw	Low value in a column.
high_value	raw	High value in a column.
density	numeric	• If **COLUMN_NAME** is available on the histogram, this column indicates the selectivity of values that span less than two endpoints in the histogram. It does not represent the selectivity of values that span two or more endpoints. • If **COLUMN_NAME** is not available on the histogram, the value of this column is 1/**NUM_DISTINCT**.
num_nulls	numeric	Number of empty values in a column.
num_buckets	numeric	Number of buckets in the histogram of a column.
last_analyzed	timestamp(0) without time zone	Date when a column was last analyzed.
sample_size	numeric	Sample size used to analyze a column.
global_stats	character varying(3)	Not supported. The value is **NULL**.
user_stats	character varying(3)	Not supported. The value is **NULL**.

Name	Type	Description
notes	character varying(99)	Not supported. The value is **NULL**.
avg_col_len	numeric	Average length of a column, in bytes.
histogram	character varying(15)	Specifies whether the histogram exists and the type of the histogram if it exists. • **NONE**: no histogram • **FREQUENCY**: frequency histogram • **EQUI-WIDTH**: equal-width histogram
scope	character varying(7)	The value **SHARED** is used to collect statistics on any table other than global temporary tables.
schema	character varying(64)	Name of the namespace to which the column belongs.

12.3.12.123 MY_TAB_COLUMNS

MY_TAB_COLUMNS displays the columns of the tables and views owned by the current user. This view exists in the PG_CATALOG and SYS schemas. All users can access this view. Only the information about the user is displayed.

Table 12-347 MY_TAB_COLUMNS columns

Name	Type	Description
owner	character varying(64)	Table owner.
table_name	character varying(64)	Table name.
column_name	character varying(64)	Column name.
data_type	character varying(128)	Data type of a column.
data_type_mod	character varying(3)	Not supported. The value is **NULL**.
data_type_owner	character varying(128)	Owner of the data type of a column.
data_length	integer	Length of a column, in bytes.

Name	Type	Description
data_precision	integer	Precision of a data type. This column is valid for the numeric data type and **NULL** for other data types.
data_scale	integer	Number of decimal places. This column is valid for the numeric data type and is set to **0** for other data types.
nullable	bpchar	Specifies whether the column can be empty (**n** for the primary key constraint and non-null constraint).
column_id	integer	Sequence number of a column when a table is created.
default_length	numeric	Length of the default value of a column, in bytes.
data_default	text	Default value of a column.
num_distinct	numeric	Number of different values in a column.
low_value	raw	Minimum value in a column.
high_value	raw	Maximum value in a column.
density	numeric	Column density.
num_nulls	numeric	Number of empty values in a column.
num_buckets	numeric	Number of buckets in the histogram of a column.
last_analyzed	timestamp(0) without time zone	Last analysis date.
sample_size	numeric	Sample size used to analyze a column.
character_set_name	character varying(44)	Not supported. The value is **NULL**.
char_col_decl_length	numeric	Declaration length of a column of the character type.
global_stats	character varying(3)	Not supported. Set it to **NO**.
user_stats	character varying(3)	Not supported. Set it to **NO**.
avg_col_len	numeric	Average length of a column, in bytes.

Name	Type	Description
char_length	numeric	Length of the column, in characters. This parameter is valid only for the varchar, nvarchar2, bpchar, and char types.
char_used	character varying(1)	Not supported. Set it to **B** if the data type is varchar, nvarchar2, bpchar, or char. For other data types, set it to **NULL**.
v80_fmt_image	character varying(3)	Not supported. The value is **NULL**.
data_upgraded	character varying(3)	Not supported. Set it to **YES**.
histogram	character varying(15)	Specifies whether the histogram exists and the type of the histogram if it exists. • **NONE**: no histogram • **FREQUENCY**: frequency histogram • **EQUI_WIDTH**: equal-width histogram
default_on_null	character varying(3)	Not supported. The value is **NULL**.
identity_column	character varying(3)	Not supported. The value is **NULL**.
sensitive_column	character varying(3)	Not supported. The value is **NULL**.
evaluation_edition	character varying(128)	Not supported. The value is **NULL**.
unusable_before	character varying(128)	Not supported. The value is **NULL**.
unusable_beginning	character varying(128)	Not supported. The value is **NULL**.
collation	character varying(100)	Collation rule of a column. This column conflicts with reserved keywords. Therefore, add the view name when calling this column.
comments	text	Comment of a column.

Name	Type	Description
schema	character varying(64)	Name of the namespace to which the column belongs.

12.3.12.124 MY_TAB_COMMENTS

MY_TAB_COMMENTS displays comments on all tables and views owned by the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-348 MY_TAB_COMMENTS columns

Name	Type	Description
owner	character varying(64)	Owner of a table or view.
table_name	character varying(64)	Name of a table or view.
comments	text	Comments.
schema	character varying(64)	Name of the namespace to which the table belongs.

12.3.12.125 MY_TAB_HISTOGRAMS

MY_TAB_HISTOGRAMS displays histogram information about the tables and views owned by the current user. All users can access this view. This view exists in the PG_CATALOG and SYS schemas.

Table 12-349 MY_TAB_HISTOGRAMS columns

Name	Type	Description
table_name	character varying(128)	Table name.
column_name	character varying(4000)	Column name or attribute of an object column.
endpoint_number	numeric	Bucket ID of the histogram.
endpoint_value	numeric	Not supported. The value is **NULL**.
endpoint_actual_value	character varying(4000)	Actual value of the bucket endpoint.
endpoint_actual_value_raw	raw	Not supported. The value is **NULL**.

Name	Type	Description
endpoint_repeat_count	numeric	Not supported. The value is **NULL**.
scope	character varying(7)	Not supported. The value is **SHARED**.

12.3.12.126 MY_TAB_MODIFICATIONS

MY_TAB_MODIFICATIONS displays statistics about modifications to tables owned by the current user since the last statistics collection on the tables. Currently, this view displays only tables on which INSERT, DELETE, and UPDATE operations have been performed. All users can access this view. This view exists in the PG_CATALOG and SYS schemas.

Table 12-350 MY_TAB_MODIFICATIONS columns

Name	Type	Description
table_name	character varying(128)	Table name.
partition_name	character varying(128)	Partition name.
subpartition_name	character varying(128)	Subpartition name.
inserts	numeric	Approximate number of insertions since the last statistics collection.
updates	numeric	Approximate number of updates since the last statistics collection.
deletes	numeric	Approximate number of deletions since the last statistics collection.
timestamp	timestamp(0) without time zone	Last modification time. Currently, the modification time of a partitioned table is not supported. The value is **NULL**.
truncated	character varying(3)	Not supported. The value is **NULL**.
drop_segments	numeric	Not supported. The value is **NULL**.
schema_name	character varying(128)	Name of the schema to which the table belongs.

12.3.12.127 MY_TAB_STATS_HISTORY

MY_TAB_STATS_HISTORY provides the table statistics history of the tables owned by the current user. All users can access this view. This view exists in the PG_CATALOG and SYS schemas.

Table 12-351 MY_TAB_STATS_HISTORY columns

Name	Type	Description
table_name	character varying(128)	Table name.
partition_name	character varying(128)	Not supported. The value is **NULL**.
subpartition_name	character varying(128)	Not supported. The value is **NULL**.
stats_update_time	timestamp(6) with time zone	Time when statistics are updated. Database restart is not supported. Otherwise, data loss will occur.

12.3.12.128 MY_TAB_STATISTICS

MY_TAB_STATISTICS displays statistics about tables owned by the current user in the database. This view exists in the PG_CATALOG and SYS schemas and all users can access this view.

Table 12-352 MY_TAB_STATISTICS columns

Name	Type	Description
table_name	character varying(128)	Table name.
partition_name	character varying(128)	Not supported. The value is **NULL**.
partition_position	numeric	Not supported. The value is **NULL**.
subpartition_name	character varying(128)	Not supported. The value is **NULL**.
subpartition_position	numeric	Not supported. The value is **NULL**.
object_type	character varying(12)	Object type. • **TABLE** • **PARTITION** • **SUBPARTITION**

Name	Type	Description
num_rows	numeric	Number of rows in an object.
blocks	numeric	Not supported. The value is **NULL**.
empty_blocks	numeric	Not supported. The value is **NULL**.
avg_space	numeric	Not supported. The value is **NULL**.
chain_cnt	numeric	Not supported. The value is **NULL**.
avg_row_len	integer	Average row length, including the row overhead.
avg_space_freelist_blocks	numeric	Not supported. The value is **NULL**.
num_freelist_blocks	numeric	Not supported. The value is **NULL**.
avg_cached_blocks	numeric	Not supported. The value is **NULL**.
avg_cache_hit_ratio	numeric	Not supported. The value is **NULL**.
im_imcu_count	numeric	Not supported. The value is **NULL**.
im_block_count	numeric	Not supported. The value is **NULL**.
im_stat_update_time	timestamp(6) without time zone	Not supported. The value is **NULL**.
scan_rate	numeric	Not supported. The value is **NULL**.
sample_size	numeric	Number of samples used for analyzing a table.
last_analyzed	timestamp with time zone	Date when a table was last analyzed. Database restart is not supported. Otherwise, data loss will occur.
global_stats	character varying(3)	Not supported. The value is **NULL**.
user_stats	character varying(3)	Not supported. The value is **NULL**.
stattype_locked	character varying(5)	Not supported. The value is **NULL**.
stale_stats	character varying(7)	Not supported. The value is **NULL**.
notes	character varying(25)	Not supported. The value is **NULL**.
scope	character varying(7)	Not supported. The default value is **SHARED**.

12.3.12.129 MY_TABLES

MY_TABLES displays information about the tables owned by the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-353 MY_TABLES columns

Name	Type	Description
owner	character varying(64)	Table owner
table_name	character varying(64)	Table name
tablespace_name	character varying(64)	Tablespace name of the table
dropped	character varying	Specifies whether the current table is deleted. ● **YES**: It is deleted. ● **NO**: It is not deleted.
num_rows	numeric	Estimated number of rows in the table
status	character varying(8)	Specifies whether the current table is valid. ● **VALID**: The current table is valid. ● **UNUSABLE**: The current table is unavailable.
sample_size	numeric	Number of samples used for analyzing the table
temporary	character(1)	Specifies whether the table is a temporary table. ● **Y**: The table is a temporary table. ● **N**: The table is not a temporary table.
pct_free	numeric	Minimum percentage of free space in a block
ini_trans	numeric	Initial number of transactions
max_trans	numeric	Maximum number of transactions
avg_row_len	integer	Average number of bytes in each row

Name	Type	Description
partitioned	character varying(3)	Specifies whether a table is a partitioned table. ● **YES**: The table is a partitioned table. ● **NO**: The table is not a partitioned table.
last_analyzed	timestamp with time zone	Last time when the table was analyzed Database restart is not supported. Otherwise, data loss will occur.
row_movement	character varying(8)	Specifies whether to allow partition row movement. ● **ENABLED**: The partition row movement is allowed. ● **DISABLED**: The partition row movement is not allowed.
compression	character varying(8)	Specifies whether to enable a table compression. ● **ENABLED**: A table compression is enabled. ● **DISABLED**: A table compression is disabled.
duration	character varying(15)	Time elapsed when a temporary table is processed. ● **NULL**: The table is not a temporary table. ● **sys$session**: The table is a temporary session table. ● **sys$transaction**: The table is a temporary transaction table.
logical_replication	character varying(8)	Specifies whether logical replication is enabled for a table. ● **ENABLED**: Logical replication is enabled. ● **DISABLED**: Logical replication is disabled.
external	character varying(3)	Specifies whether the table is an external table. ● **YES**: The table is an external table. ● **NO**: The table is not an external table.

Name	Type	Description
logging	character varying(3)	Specifies whether to record logs for table changes. • **YES**: Logs are recorded for table changes. • **NO**: Logs are not recorded for table changes.
default_collation	character varying(100)	Default collation of a table
degree	character varying(10)	Number of instances in a scanned table
table_lock	character varying(8)	Specifies whether to enable a table lock. • **ENABLED**: The table lock is enabled. • **DISABLED**: The table lock is disabled.
nested	character varying(3)	Specifies whether a table is a nested table. • **YES**: The table is a nested table. • **NO**: The table is not a nested table.
buffer_pool	character varying(7)	Default buffer pool of a table
flash_cache	character varying(7)	Not supported. The value is **NULL**.
cell_flash_cache	character varying(7)	Not supported. The value is **NULL**.
skip_corrupt	character varying(8)	Specifies whether to skip corrupted blocks during table scanning. • **ENABLED**: The corrupted block is skipped. • **DISABLED**: The corrupted block is not skipped.
has_identity	character varying(3)	Specifies whether a table has an identifier column. • **YES**: There is an identifier column. • **NO**: There is no identifier column.

Name	Type	Description
segment_created	character varying(3)	Specifies whether a table segment has been created. • **YES**: The table segment has been created. • **NO**: The table segment is not created.
monitoring	character varying(3)	Specifies whether to monitor the modification of a table. • **YES**: The modification of the table is monitored. • **NO**: The modification of the table is not monitored.
cluster_name	character varying(128)	Not supported. The value is **NULL**.
iot_name	character varying(128)	Not supported. The value is **NULL**.
pct_used	numeric	Not supported. The value is **NULL**.
initial_extent	numeric	Not supported. The value is **NULL**.
next_extent	numeric	Not supported. The value is **NULL**.
min_extents	numeric	Not supported. The value is **NULL**.
max_extents	numeric	Not supported. The value is **NULL**.
pct_increase	numeric	Not supported. The value is **NULL**.
freelists	numeric	Not supported. The value is **NULL**.
freelist_groups	numeric	Not supported. The value is **NULL**.
backed_up	character varying(1)	Not supported. The value is **NULL**.
blocks	numeric	Not supported. The value is **NULL**.
empty_blocks	numeric	Not supported. The value is **NULL**.

Name	Type	Description
avg_space	numeric	Not supported. The value is **NULL**.
chain_cnt	numeric	Not supported. The value is **NULL**.
avg_space_freelist_blocks	numeric	Not supported. The value is **NULL**.
num_freelist_blocks	numeric	Not supported. The value is **NULL**.
instances	character varying(10)	Not supported. The value is **NULL**.
cache	character varying(5)	Not supported. The value is **NULL**.
iot_type	character varying(12)	Not supported. The value is **NULL**.
secondary	character varying(1)	Not supported. The value is **NULL**.
global_stats	character varying(3)	Not supported. The value is **NULL**.
user_stats	character varying(3)	Not supported. The value is **NULL**.
cluster_owner	character varying(30)	Not supported. The value is **NULL**.
dependencies	character varying(8)	Not supported. The value is **NULL**.
compression_for	character varying(30)	Not supported. The value is **NULL**.
read_only	character varying(3)	Not supported. The value is **NULL**.
result_cache	character varying(7)	Not supported. The value is **NULL**.
clustering	character varying(3)	Not supported. The value is **NULL**.
activity_tracking	character varying(23)	Not supported. The value is **NULL**.
dml_timestamp	character varying(25)	Not supported. The value is **NULL**.
container_data	character varying(3)	Not supported. The value is **NULL**.

Name	Type	Description
inmemory_priority	character varying(8)	Not supported. The value is **NULL**.
inmemory_distribute	character varying(15)	Not supported. The value is **NULL**.
inmemory_compression	character varying(17)	Not supported. The value is **NULL**.
inmemory_duplicate	character varying(13)	Not supported. The value is **NULL**.
duplicated	character varying(1)	Not supported. The value is **NULL**.
sharded	character varying(1)	Not supported. The value is **NULL**.
hybrid	character varying(3)	Not supported. The value is **NULL**.
cellmemory	character varying(24)	Not supported. The value is **NULL**.
containers_default	character varying(3)	Not supported. The value is **NULL**.
container_map	character varying(3)	Not supported. The value is **NULL**.
extended_data_link	character varying(3)	Not supported. The value is **NULL**.
extended_data_link_map	character varying(3)	Not supported. The value is **NULL**.
inmemory_service	character varying(12)	Not supported. The value is **NULL**.
inmemory_service_name	character varying(1000)	Not supported. The value is **NULL**.
container_map_object	character varying(3)	Not supported. The value is **NULL**.
memoptimize_read	character varying(8)	Not supported. The value is **NULL**.
memoptimize_write	character varying(8)	Not supported. The value is **NULL**.
has_sensitive_column	character varying(3)	Not supported. The value is **NULL**.
admit_null	character varying(3)	Not supported. The value is **NULL**.

Name	Type	Description
data_link_dml_enabled	character varying(3)	Not supported. Set it to null.
object_id_type	character varying(16)	Not supported. The value is **NULL**.
table_type_owner	character varying(128)	Not supported. The value is **NULL**.
table_type	character varying(128)	Not supported. The value is **NULL**.
compress_for	character varying(30)	Not supported. The value is **NULL**.

12.3.12.130 MY_TABLESPACES

MY_TABLESPACES displays the description of tablespaces that store objects owned by users. By default, it is accessible to all users. This view exists in the PG_CATALOG and SYS schemas. The logical structure features of the GaussDB are different from those of the A database.

Table 12-354 MY_TABLESPACES columns

Name	Type	Description
tablespace_name	character varying(64)	Tablespace name.
block_size	numeric	Not supported. The value is **NULL**.
initial_extent	numeric	Not supported. The value is **NULL**.
next_extent	numeric	Not supported. The value is **NULL**.
min_extents	numeric	Not supported. The value is **NULL**.
max_extents	numeric	Not supported. The value is **NULL**.
max_size	numeric	Not supported. The value is **NULL**.
pct_increase	numeric	Not supported. The value is **NULL**.
min_extlen	numeric	Not supported. The value is **NULL**.

Name	Type	Description
contents	character varying(9)	Not supported. The value is **NULL**.
status	character varying(9)	Tablespace status. The default value is **ONLINE**.
logging	character varying(9)	Not supported. The value is **NULL**.
force_logging	character varying(3)	Not supported. The value is **NULL**.
extent_management	character varying(10)	Not supported. The value is **NULL**.
allocation_type	character varying(9)	Not supported. The value is **NULL**.
segment_space_management	character varying(6)	Not supported. The value is **NULL**.
def_tab_compression	character varying(8)	Not supported. The value is **NULL**.
retention	character varying(11)	Not supported. The value is **NULL**.
bigfile	character varying(3)	Not supported. The value is **NULL**.
predicate_evaluation	character varying(7)	Not supported. The value is **NULL**.
encrypted	character varying(3)	Not supported. The value is **NULL**.
compress_for	character varying(30)	Not supported. The value is **NULL**.
def_inmemory	character varying(8)	Not supported. The value is **NULL**.
def_inmemory_priority	character varying(8)	Not supported. The value is **NULL**.
def_inmemory_distribute	character varying(15)	Not supported. The value is **NULL**.
def_inmemory_compression	character varying(17)	Not supported. The value is **NULL**.
def_inmemory_duplicate	character varying(13)	Not supported. The value is **NULL**.
shared	character varying(12)	Not supported. The value is **NULL**.

Name	Type	Description
def_index_compression	character varying(8)	Not supported. The value is **NULL**.
index_compress_for	character varying(13)	Not supported. The value is **NULL**.
def_cellmemory	character varying(14)	Not supported. The value is **NULL**.
def_inmemory_service	character varying(12)	Not supported. The value is **NULL**.
def_inmemory_service_name	character varying(1000)	Not supported. The value is **NULL**.
lost_write_protect	character varying(7)	Not supported. The value is **NULL**.
chunk_tablespace	character varying(1)	Not supported. The value is **NULL**.

12.3.12.131 MY_TRIGGERS

MY_TRIGGERS displays information about the triggers owned by the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-355 MY_TRIGGERS columns

Name	Type	Description
owner	character varying(128)	Trigger owner.
trigger_name	character varying(64)	Trigger name.
trigger_type	character varying	Trigger type, which can be **before statement**, **before each row**, **after statement**, **after each row**, or **instead of**.
triggering_event	character varying	Event that triggers a trigger. The value can be **update**, **insert**, **delete**, or **truncate**.
table_owner	character varying(64)	Owner of the table that defines a trigger.
base_object_type	character varying(18)	Basic object type of a trigger, which can be **table** or **view**.
table_name	character varying(64)	Name of the table or view that defines a trigger.

Name	Type	Description
column_name	character varying(4000)	Not supported. The value is **NULL**.
referencing_name	character varying(422)	Not supported. Set it to **referencing new as new old as old**.
when_clause	character varying(4000)	Content of **when**. **TRUE** must be evaluated as **TRIGGER_BODY** to execute.
status	character varying(64)	• **O**: The trigger is enabled in origin or local mode. • **D**: The trigger is disabled. • **R**: The trigger is enabled in replica mode. • **A**: The trigger is always enabled.
description	character varying(4000)	Trigger description, which is used to rebuild a trigger creation statement.
action_type	character varying(11)	Action type of a trigger, which only supports **call**.
trigger_body	text	Statement executed when a trigger is triggered.
crossedition	character varying(7)	Not supported. The value is **NULL**.
before_statement	character varying(3)	Not supported. The value is **NULL**.
before_row	character varying(3)	Not supported. The value is **NULL**.
after_row	character varying(3)	Not supported. The value is **NULL**.
after_statement	character varying(3)	Not supported. The value is **NULL**.
instead_of_row	character varying(3)	Not supported. The value is **NULL**.
fire_once	character varying(3)	Not supported. The value is **NULL**.
apply_server_only	character varying(3)	Not supported. The value is **NULL**.

12.3.12.132 MY_TYPE_ATTRS

MY_TYPE_ATTRS displays all types of attributes owned by the current user in the database. This view exists in the PG_CATALOG and SYS schemas and all users can access this view.

Table 12-356 MY_TYPE_ATTRS columns

Name	Type	Description
type_name	character varying(128)	Data type name.
attr_name	character varying(128)	Attribute name.
attr_type_mod	character varying(7)	Type modifier of an attribute: • **REF** • **POINT**
attr_type_owner	character varying(128)	Owner of an attribute type.
attr_type_name	character varying(128)	Name of an attribute type.
length	numeric	Length of the CHAR attribute, or the maximum length of the VARCHAR and character varying attribute.
precision	numeric	Decimal precision of a number or DECIMAL attribute, or binary precision of a FLOAT attribute.
scale	numeric	Decimal places for a numeric or DECIMAL attribute.
character_set_name	character varying(44)	Character set name (**Char_CS** or **NCHAR_CS**) of an attribute.
attr_no	numeric	Syntax order number or location (not used as an ID number) of an attribute specified in the type specification or CREATE TYPE statement.
inherited	character varying(3)	Specifies whether an attribute is inherited from a supertype. • **YES**: It is inherited from a supertype. • **NO**: It is not inherited from a supertype.

12.3.12.133 MY_TYPES

MY_TYPES describes all object types owned by the current user. All users can access this view. This view exists in the PG_CATALOG and SYS schemas.

Table 12-357 MY_TYPES columns

Name	Type	Description
type_name	character varying(128)	Type name.
type_oid	raw	Type OID.
typecode	character varying(128)	Type code.
attributes	numeric	Number of attributes in a type.
methods	numeric	Not supported. The value is **0**.
predefined	character varying(3)	Specifies whether the type is a predefined type.
incomplete	character varying(3)	Specifies whether the type is incomplete.
final	character varying(3)	Not supported. The value is **NULL**.
instantiable	character varying(3)	Not supported. The value is **NULL**.
persistable	character varying(3)	Not supported. The value is **NULL**.
supertype_owner	character varying(128)	Not supported. The value is **NULL**.
supertype_name	character varying(128)	Not supported. The value is **NULL**.
local_attributes	numeric	Not supported. The value is **NULL**.
local_methods	numeric	Not supported. The value is **NULL**.
typeid	raw	Not supported. The value is **NULL**.

12.3.12.134 MY_VIEWS

MY_VIEWS displays information about all views of the current user. This view exists in the PG_CATALOG and SYS schemas.

Table 12-358 MY_VIEWS columns

Name	Type	Description
owner	character varying(64)	Owner of a view.

Name	Type	Description
view_name	character varying(64)	Name of the view.
text	text	Text of a view.
text_length	integer	Text length of a view.
text_vc	character varying(4000)	View creation statement. This column may truncate the view text. The BEQUEATH clause will not appear as part of the **TEXT_VC** column in this view.
type_text_length	numeric	Not supported. The value is **NULL**.
type_text	character varying(4000)	Not supported. The value is **NULL**.
oid_text_length	numeric	Not supported. The value is **NULL**.
oid_text	character varying(4000)	Not supported. The value is **NULL**.
view_type_owner	character varying(128)	Not supported. The value is **NULL**.
view_type	character varying(128)	Not supported. The value is **NULL**.
superview_name	character varying(128)	Not supported. The value is **NULL**.
editioning_view	character varying(1)	Not supported. The value is **NULL**.
read_only	character varying(1)	Not supported. The value is **NULL**.
container_data	character varying(1)	Not supported. The value is **NULL**.
bequeath	character varying(12)	Not supported. The value is **NULL**.
origin_con_id	character varying(256)	Not supported. The value is **NULL**.
default_collation	character varying(100)	Not supported. The value is **NULL**.
containers_default	character varying(3)	Not supported. The value is **NULL**.

Name	Type	Description
container_map	character varying(3)	Not supported. The value is **NULL**.
extended_data_link	character varying(3)	Not supported. The value is **NULL**.
extended_data_link_map	character varying(3)	Not supported. The value is **NULL**.
has_sensitive_column	character varying(3)	Not supported. The value is **NULL**.
admit_null	character varying(3)	Not supported. The value is **NULL**.
pdb_local_only	character varying(3)	Not supported. The value is **NULL**.

12.3.12.135 NLS_DATABASE_PARAMETERS

NLS_DATABASE_PARAMETERS lists the permanent NLS parameters of the database server. This view exists in the PG_CATALOG and SYS schemas. It is accessible to all users. Due to different database kernels and parameter setting formats, the query results of the same parameters in the database may be obviously different from those in the database A.

Table 12-359 NLS_DATABASE_PARAMETERS columns

Name	Type	Description
parameter	character varying(128)	Parameter name
value	character varying(64)	Parameter value

12.3.12.136 NLS_INSTANCE_PARAMETERS

NLS_INSTANCE_PARAMETERS lists the permanent NLS parameters of the database client. This view exists in the PG_CATALOG and SYS schemas. It is accessible to all users. Due to different database kernels and parameter setting formats, the query results of the same parameters in the database may be obviously different from those in the database A.

Table 12-360 NLS_INSTANCE_PARAMETERS columns

Name	Type	Description
parameter	character varying(128)	Parameter name

Name	Type	Description
value	character varying(64)	Parameter value

12.3.12.137 PG_AVAILABLE_EXTENSION_VERSIONS

PG_AVAILABLE_EXTENSION_VERSIONS displays extension versions of certain database features. This view is for internal use only. You are advised not to use it.

Table 12-361 PG_AVAILABLE_EXTENSION_VERSIONS columns

Name	Type	Description
name	name	Extension name
version	text	Version name
installed	boolean	Specifies whether this extension version is installed.
superuser	boolean	Specifies whether only system administrators are allowed to install the extension.
relocatable	boolean	Specifies whether the extension can be relocated to another schema.
schema	name	Name of the schema that the extension must be installed into (**NULL** if the extension is partially or fully relocatable)
requires	name[]	Names of prerequisite extensions (**NULL** if none)
comment	text	Comment from the extension's control file.

12.3.12.138 PG_AVAILABLE_EXTENSIONS

PG_AVAILABLE_EXTENSIONS displays the extension information about certain database features. This view is for internal use only. You are advised not to use it.

Table 12-362 PG_AVAILABLE_EXTENSIONS columns

Name	Type	Description
name	name	Extension name
default_version	text	Name of the default version (**NULL** if none is specified)
installed_version	text	Currently installed version of the extension (**NULL** if no version is installed)

Name	Type	Description
comment	text	Comment from the extension's control file.

12.3.12.139 PG_CONTROL_GROUP_CONFIG

PG_CONTROL_GROUP_CONFIG stores Cgroup configuration information in the system. Only the user with sysadmin permission can query this view.

Table 12-363 PG_CONTROL_GROUP_CONFIG columns

Name	Type	Description
pg_control_group_config	text	Configuration information of the Cgroup

12.3.12.140 PG_CURSORS

PG_CURSORS displays cursors that are currently available.

Table 12-364 PG_CURSORS columns

Name	Type	Description
name	text	Cursor name
statement	text	Query statement when the cursor is declared to change
is_holdable	boolean	**True** if the cursor is holdable (it can be accessed after the transaction that declared the cursor has committed); **false** otherwise
is_binary	boolean	Whether the cursor was declared BINARY. If it was, the value is **TRUE**.
is_scrollable	boolean	Whether the cursor is scrollable (it allows rows to be retrieved in a nonsequential manner). If it is, the value is **TRUE**.
creation_time	timestamp with time zone	Timestamp at which the cursor is declared

12.3.12.141 PG_EXT_STATS

PG_EXT_STATS displays extended statistics stored in **PG_STATISTIC_EXT**. The extension statistics means multiple columns of statistics.

Table 12-365 PG_EXT_STATS columns

Name	Type	Reference	Description
schemaname	name	PG_NAMESPACE.nspname	Name of the schema that contains a table.
tablename	name	PG_CLASS.relname	Table name.
attname	int2vector	PG_STATISTIC_EXT.stakey	Columns to be combined for collecting statistics
inherited	Boolean	-	Inherited tables are not supported currently. The value of this column is **false**.
null_frac	real	-	Percentage of column combinations that are null to all records
avg_width	integer	-	Average width of column combinations, in byte
n_distinct	real	-	• Estimated number of distinct values in a column combination if the value is greater than 0 • Negative number obtained by multiplying the result calculated by dividing the number of distinct values by the number of rows by **–1** if the value is less than **0**. For example, **–1** indicates that the number of distinct values is the same as the number of rows for a column combination. 1. The negated form is used when ANALYZE believes that the number of distinct values is likely to increase as the table grows. 2. The positive form is used when the column seems to have a fixed number of possible values. • The number of distinct values is unknown if the value is **0**.

Name	Type	Reference	Description
n_dndistinct	real	-	Number of not-null distinct values in the **dn1** column combination. ● Exact number of distinct values if the value is greater than 0. ● Negative number obtained by multiplying the result calculated by dividing the number of distinct values by the number of rows by –1 if the value is less than 0. For example, if a value in a column combination appears twice in average, **n_dndistinct** equals **–0.5**. ● The number of distinct values is unknown if the value is **0**.
most_common_vals	anyarray	-	List of the most common values in a column combination. If this combination does not have the most common values, the column will be **NULL**. None of the most common values in the column is **NULL**.
most_common_freqs	real[]	-	List of the frequencies of the most common values in a column combination. The frequencies are obtained by dividing the number of occurrences of each value by the number of rows. If the value of **most_common_vals** is **NULL**, the value of this column is **NULL**.
most_common_vals_null	anyarray	-	List of the most common values in a column combination. If this combination does not have the most common values, the column will be **NULL**. At least one of the common values in the column is **NULL**.

Name	Type	Reference	Description
most_common_freqs_null	real[]	-	List of the frequencies of the most common values in a column combination. The frequencies are obtained by dividing the number of occurrences of each value by the number of rows. If the value of **most_common_vals_null** is **NULL**, the value of this column is **NULL**.
histogram_bounds	anyarray	-	Boundary value list of the histogram
partitionname	name	PG_PARTITION.relname	Name of the level-1 partition in the partitioned table. For a non-partitioned table, this field is left blank.
subpartitionname	name	PG_PARTITION.relname	Name of the level-2 partition in the partitioned table. For a non-partitioned table or level-1 partitioned table, this field is left blank.

12.3.12.142 PG_GET_SENDERS_CATCHUP_TIME

PG_GET_SENDERS_CATCHUP_TIME provides catchup information of the currently active primary/standby instance sender thread on the database node.

Table 12-366 PG_GET_SENDERS_CATCHUP_TIME columns

Name	Type	Description
pid	bigint	Current sender thread ID
lwpid	integer	Current sender lwpid
local_role	text	Local role
peer_role	text	Peer role

Name	Type	Description
state	text	Current sender's replication status • **Startup**: startup state. • **Backup**: backup state. • **Catchup**: catch-up state, which indicates that the standby node is catching up with the primary node. • **Streaming**: streaming replication state. When the standby node catches up with the primary node, the replication remains in this state.
type	text	Current sender type. • **Wal**: type that logs are written in advance. • **Data**: data type.
catchup_start	timestamp with time zone	Startup time of a catchup task
catchup_end	timestamp with time zone	End time of a catchup task

12.3.12.143 PG_GROUP

PG_GROUP displays the database role authentication and the relationship between roles.

Table 12-367 PG_GROUP columns

Name	Type	Description
groname	name	Group name
grosysid	oid	Group ID
grolist	oid[]	An array, including all the role IDs in this group

12.3.12.144 PG_GTT_RELSTATS

PG_GTT_RELSTATS displays basic information about all global temporary tables of the current session by invoking **pg_get_gtt_relstats**.

Table 12-368 PG_GTT_RELSTATS columns

Name	Type	Description
schemaname	name	Schema name
tablename	name	Name of a global temporary table
relfilenode	oid	ID of a file object
relpages	integer	Number of disk pages of a global temporary table
reltuples	real	Number of records in a global temporary table
relallvisible	integer	Number of pages that are marked as all visible
relfrozenxid	xid	All transaction IDs before this one have been replaced with a permanent (frozen) transaction ID in the table.
relminmxid	xid	Reserved

12.3.12.145 PG_GTT_STATS

PG_GTT_STATS displays statistics about a single column in all global temporary tables of the current session by calling **pg_get_gtt_relstats**.

Table 12-369 PG_GTT_STATS columns

Name	Type	Description
schemaname	name	Schema name.
tablename	name	Name of a global temporary table.
attname	name	Attribute name.
inherited	boolean	Specifies whether to collect statistics for objects that have inheritance relationship.
null_frac	real	Percentage of column entries that are null.
avg_width	integer	Average stored width, in bytes, of non-null entries.
n_distinct	real	Number of distinct, non-null data values in the column.
most_common_vals	text[]	List of the most common values, which is sorted by occurrence frequency.
most_common_freqs	real[]	Frequencies of the most common values.

Name	Type	Description
histogram_bounds	text[]	Data distribution (excluding the most common values) in a frequency histogram description column.
correlation	real	Correlation coefficient.
most_common_elems	text[]	List of the most common element values, which is used for the array type or some other type.
most_common_elem_freqs	real[]	Frequencies of the most common element values.
elem_count_histogram	real[]	Array type histogram.

12.3.12.146 PG_GTT_ATTACHED_PIDS

PG_GTT_ATTACHED_PIDS checks which sessions are using global temporary tables by calling the **pg_get_attached_pid** function.

Table 12-370 PG_GTT_ATTACHED_PIDS columns

Name	Type	Description
schemaname	name	Schema name
tablename	name	Name of a global temporary table
relid	oid	OID of a global temporary table
pids	bigint[]	Thread PID list
sessionids	bigint[]	Session ID list

12.3.12.147 PG_INDEXES

PG_INDEXES provides access to useful information about each index in the database.

Table 12-371 PG_INDEXES columns

Name	Type	Reference	Description
schemaname	name	PG_NAMESPACE.nspname	Name of the schema that contains tables and indexes
tablename	name	PG_CLASS.relname	Name of the table for which the index serves

Name	Type	Reference	Description
indexname	name	PG_CLASS.relname	Index name
tablespace	name	PG_TABLESPACE.nspname	Name of the tablespace that contains the index
indexdef	text	-	Index definition (a reconstructed **CREATE INDEX** command)

12.3.12.148 PG_LOCKS

PG_LOCKS displays information about locks held by open transactions.

Table 12-372 PG_LOCKS columns

Name	Type	Reference	Description
locktype	text	-	Type of the locked object: relation, extend, page, tuple, transactionid, virtualxid, object, userlock, and advisory
database	oid	OID in PG_DATABASE	OID of the database in which the locked target exists • The OID is **0** if the target is a shared object. • The OID is **NULL** if the locked target is a transaction.
relation	oid	OID in PG_CLASS	OID of the relationship targeted by the lock (**NULL** if the object is not a relation or part of a relation)
page	integer	-	Page number targeted by the lock within the relation (**NULL** if the object is not a relation page or row page)
tuple	smallint	-	Row number targeted by the lock within the page (**NULL** if the object is not a row)
bucket	integer	-	Bucket number corresponding to the child table. The value is **NULL** if the target is not a table.
virtualxid	text	-	Virtual ID of the transaction targeted by the lock (**NULL** if the object is not a virtual transaction)

Name	Type	Reference	Description
transactionid	xid	-	ID of the transaction targeted by the lock (**NULL** if the object is not a transaction)
classid	oid	OID in **PG_CLASS**	OID of the system catalog that contains the object (**NULL** if the object is not a general database object)
objid	oid	-	OID of the lock target within its system table (**NULL** if the target is not a general database object)
objsubid	smallint	-	Column number for a column in the table (**0** if the object is of other object type and **NULL** if the object is not a general database object)
virtualtransaction	text	-	Virtual ID of the transaction holding or awaiting this lock
pid	bigint	-	Logical ID of the server thread holding or awaiting this lock (**NULL** if the lock is held by a prepared transaction)
sessionid	bigint	-	ID of the session that holds or waits for the lock
mode	text	-	Lock mode held or desired by this thread The value can be **AccessShareLock**, **RowShareLock**, **RowExclusiveLock**, **ShareLock**, **ShareRowExclusive-Lock**, **ExclusiveLock**, or **AccessExclusiveLock**.
granted	Boolean	-	• The value is **TRUE** if the lock is a held lock. • The value is **FALSE** if the lock is an awaited lock.
fastpath	Boolean	-	The value is **TRUE** if the lock is obtained through **fast-path**, and is **FALSE** if the lock is obtained through the main lock table.
locktag	text	-	Lock information that the session waits for. It can be parsed using the **locktag_decode()** function.
global_sessionid	text	-	Global session ID

12.3.12.149 PG_NODE_ENV

PG_NODE_ENV obtains environmental variable information about the current node.

Table 12-373 PG_NODE_ENV columns

Name	Type	Description
node_name	text	Current node name
host	text	Host name of the node
process	integer	Number of the node process
port	integer	Port ID of the node
installpath	text	Installation directory of the node
datapath	text	Data directory of the node
log_directory	text	Log directory of the node

12.3.12.150 PG_OS_THREADS

PG_OS_THREADS provides status information about all the threads under the current node.

Table 12-374 PG_OS_THREADS columns

Name	Type	Description
node_name	text	Current node name
pid	bigint	PID of the thread running under the current node process
lwpid	integer	Lightweight thread ID corresponding to the PID
thread_name	text	Thread name corresponding to the PID
creation_time	timestamp with time zone	Thread creation time corresponding to the PID

12.3.12.151 PG_PREPARED_STATEMENTS

PG_PREPARED_STATEMENTS displays all prepared statements that are available in the current session.

Table 12-375 PG_PREPARED_STATEMENTS columns

Name	Type	Description
name	text	Identifier of the prepared statement
statement	text	Query string for creating this prepared statement. For prepared statements created through SQL, this is the PREPARE statement submitted by the client. For prepared statements created through the frontend/backend protocol, this is the text of the prepared statement itself.
prepare_time	timestamp with time zone	Timestamp when the prepared statement is created
parameter_types	regtype[]	Expected parameter types for the prepared statement in the form of an array of **regtype**. The OID corresponding to an element of this array can be obtained by converting the **regtype** value to **oid**.
from_sql	boolean	• **True** if the prepared statement was created through the PREPARE statement • **False** if the statement was prepared through the frontend/backend protocol

12.3.12.152 PG_PREPARED_XACTS

PG_PREPARED_XACTS displays information about transactions that are currently prepared for two-phase commit.

Table 12-376 PG_PREPARED_XACTS columns

Name	Type	Reference	Description
transaction	xid	-	Numeric transaction identifier of the prepared transaction
gid	text	-	Global transaction identifier that was assigned to the transaction
prepared	timestamp with time zone	-	Time at which the transaction is prepared for commit
owner	name	PG_AUTHID.rolname	Name of the user that executes the transaction

Name	Type	Reference	Description
database	name	PG_DATABASE.datname	Name of the database in which the transaction is executed

12.3.12.153 PG_REPLICATION_ORIGIN_STATUS

PG_REPLICATION_ORIGIN_STATUS displays the replication status of the replication source.

Table 12-377 PG_REPLICATION_ORIGIN_STATUS columns

Name	Type	Description
local_id	oid	Replication source ID.
external_id	text	Name of the replication source.
remote_lsn	text	Remote LSN of the replication source.
local_lsn	text	Local LSN of the replication source.

12.3.12.154 PG_REPLICATION_SLOTS

PG_REPLICATION_SLOTS displays replication slot information.

Table 12-378 PG_REPLICATION_SLOTS columns

Name	Type	Description
slot_name	text	Replication slot name.
plugin	text	Name of the output plug-in corresponding to the logical replication slot.
slot_type	text	Replication slot type. • **physical**: physical replication slot. • **logical**: logical replication slot.
datoid	oid	OID of the database where the replication slot resides.
database	name	Name of the database where the replication slot resides.

Name	Type	Description
active	Boolean	Determines whether the replication slot is activated. • **t** (true): yes • **f** (false): no
xmin	xid	XID of the earliest transaction that the database must reserve for the replication slot.
catalog_xmin	xid	XID of the earliest system catalog-involved transaction that the database must reserve for the logical replication slot.
restart_lsn	text	Physical location of the earliest Xlog required by the replication slot.
dummy_standby	Boolean	Reserved parameter.
confirmed_flush	text	Dedicated logical replication slot. The client confirms the location of the received log.
confirmed_csn	xid	Dedicated logical replication slot. The client confirms the CSN corresponding to the last transaction in the received log.

Example:

```
gaussdb=# SELECT * FROM pg_replication_slots;
 slot_name |      plugin      | slot_type | datoid | database | active | xmin | catalog_xmin | restart_lsn |
dummy_standby | confirmed_flush | confirmed_csn
-----------+------------------+-----------+--------+----------+--------+------+--------------+-------------+---------------
+-----------------+---------------
 dn_6002   |                  | physical  |      0 |          | t      |      |              | 0/3622B528  | f             |               |
 dn_6003   |                  | physical  |      0 |          | t      |      |              | 0/3622B528  | f             |               |
 slot_lsn  | mppdb_decoding | logical   | 131072 | db_test  | f      |      |        66658 | 0/36252350  | f             |
0/362523D0       |
 slot_test | mppdb_decoding | logical   | 131072 | db_test  | f      |      |        66658 | 0/36251718  | f
|               |      10025527
(4 rows)
```

NOTICE

When a query is performed on a DN, the confirmed_csn query result of the LSN-based logical replication slot is empty, and the confirmed_flush query result of the CSN-based logical replication slot is empty.

12.3.12.155 PG_RULES

PG_RULES provides access to query useful information about rewrite rules.

Table 12-379 PG_RULES columns

Name	Type	Description
schemaname	name	Name of the schema that contains a table
tablename	name	Name of the table to which the rule applies
rulename	name	Name of a rule
definition	text	Rule definition (a reconstructed creation command)

12.3.12.156 PG_RUNNING_XACTS

PG_RUNNING_XACTS displays the running transaction information on the current node.

Table 12-380 PG_RUNNING_XACTS columns

Name	Type	Description
handle	integer	Slot handle in the transaction manager corresponding to a transaction. The value is fixed at **–1**.
gxid	xid	Transaction ID
state	tinyint	Transaction status (**3**: prepared or **0**: starting)
node	text	Node name
xmin	xid	Minimum transaction ID on the node
vacuum	boolean	Whether the current transaction is lazy vacuum • **t** (true): yes • **f** (false): no
timeline	bigint	Number of database restarts
prepare_xid	xid	Transaction ID in the **prepared** state. If the status is not **prepared**, the value is **0**.
pid	bigint	Thread ID corresponding to the transaction
next_xid	xid	Transaction ID sent by other nodes to the current node. The value is fixed at **0**.

12.3.12.157 PG_SETTINGS

PG_SETTINGS displays information about parameters of the running database.

Table 12-381 PG_SETTINGS columns

Name	Type	Description
name	text	Parameter name
setting	text	Current parameter value
unit	text	Implicit unit of the parameter
category	text	Logical group of the parameter
short_desc	text	Brief description of the parameter
extra_desc	text	Detailed description of the parameter
context	text	Context required to set the parameter value, including **internal**, **postmaster**, **sighup**, **backend**, **superuser**, and **user**
vartype	text	Parameter type, including **bool**, **enum**, **integer**, **real**, or **string**
source	text	Method of assigning the parameter value
min_val	text	Minimum value of the parameter. If the parameter type is not numeric, the value of this column is **null**.
max_val	text	Maximum value of the parameter. If the parameter type is not numeric, the value of this column is **null**.
enumvals	text[]	Valid values of an enum-type parameter. If the parameter type is not enum, the value of this column is **null**.
boot_val	text	Default parameter value used upon the database startup
reset_val	text	Default parameter value used upon the database reset
sourcefile	text	Configuration file used to set parameter values. If parameter values are not configured using the configuration file, the value of this column is **null**.
sourceline	integer	Row number of the configuration file for setting parameter values. If parameter values are not configured using the configuration file, the value of this column is **null**.

12.3.12.158 PG_STATS

PG_STATS displays single-column statistics stored in the **pg_statistic** table. The **autovacuum_naptime** parameter specifies the interval for updating statistics recorded in the view.

Table 12-382 PG_STATS columns

Name	Type	Reference	Description
schemaname	name	PG_NAMESPACE.nspname	Name of the schema that contains a table.
tablename	name	PG_CLASS.relname	Table name.
attname	name	PG_ATTRIBUTE.attname	Column name
inherited	Boolean	-	Inherited tables are not supported currently. The value of this column is **false**.
null_frac	real	-	Percentage of column entries that are null
avg_width	integer	-	Average width in bytes of column's entries
n_distinct	real	-	• Estimated number of distinct values in the column if the value is greater than 0 • Negative number obtained by multiplying the result calculated by dividing the number of distinct values by the number of rows by –1 if the value is less than 0. For example, value **–1** indicates that the number of distinct values is the same as the number of rows for a unique column. 1. The negated form is used when ANALYZE believes that the number of distinct values is likely to increase as the table grows. 2. The positive form is used when the column seems to have a fixed number of possible values. • The number of distinct values is unknown if the value is **0**.

Name	Type	Reference	Description
n_dndistinct	real	-	Number of not-null distinct values in the **dn1** column. ● Exact number of distinct values if the value is greater than **0**. ● Negative number obtained by multiplying the result calculated by dividing the number of distinct values by the number of rows by –1 if the value is less than 0. For example, if the value of a column appears twice in average, set **n_dndistinct=-0.5**. ● The number of distinct values is unknown if the value is **0**.
most_common_vals	anyarray	-	List of the most common values in a column. If this column does not have the most common value, the value is **NULL**.
most_common_freqs	real[]	-	List of the frequencies of the most common values in a column. The frequencies are obtained by dividing the number of occurrences of each value by the number of rows. The value is **NULL** if the value of **most_common_vals** is **NULL**.
histogram_bounds	anyarray	-	Frequency histogram consisting of values excluding null values and MVC values. If a value appears in the value of **most_common_vals**, it does not appear in the histogram. If the column data type does not have the < operator or the list specified by **most_common_vals** contains all values of the column, the histogram information of the column is **NULL**.

Name	Type	Reference	Description
correlation	real	-	Correlation between the physical row sequence and logical row sequence of a column value. The value ranges from –1 to +1. When the value is close to –1 or +1, the index scan overhead is less than that when the value is close to 0 because random access to the disk is reduced. This column is **NULL** if the column data type does not have a < operator.
most_common_elems	anyarray	-	A list of non-null element values most often appearing
most_common_elem_freqs	real[]	-	List that records the frequency of the most commonly used non-null elements.
elem_count_histogram	real[]	-	A histogram of the counts of distinct non-null element values
partitionname	name	PG_PARTITION.relname	Name of the level-1 partition in the partitioned table. For a non-partitioned table, this field is left blank.
subpartitionname	name	PG_PARTITION.relname	Name of the level-2 partition in the partitioned table. For a non-partitioned table or level-1 partitioned table, this field is left blank.

12.3.12.159 PG_STAT_ACTIVITY

PG_STAT_ACTIVITY displays information about the current user's queries. The columns save information about the last query.

Table 12-383 PG_STAT_ACTIVITY columns

Name	Type	Description
datid	oid	OID of the database that the user session connects to in the backend.
datname	name	Name of the database that the user session connects to in the backend.
pid	bigint	ID of the backend thread.

Name	Type	Description
sessionid	bigint	Session ID.
usesysid	oid	OID of the user logged in to the backend.
usename	name	Name of the user logged in to the backend.
application_name	text	Name of the application connected to the backend.
client_addr	inet	IP address of the client connected to the backend. If this column is **NULL**, it indicates either the client is connected through a Unix socket on the server or this is an internal process, such as autovacuum.
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of **client_addr**. This column will be non-null only for IP connections and only when **log_hostname** is enabled.
client_port	integer	TCP port number that the client uses for communication with the backend (**–1** if a Unix socket is used)
backend_start	timestamp with time zone	Time when this session was started, that is, when the client connected to the server.
xact_start	timestamp with time zone	Time when the current active transaction was started (**NULL** if no transaction is active). If the current query is the first of its transaction, the value of this column is the same as that of the **query_start** column.
query_start	timestamp with time zone	Time when the current active query was started, or time when the last query was started if the value of **state** is not **active**. For a stored procedure, function, or package, the first query time is displayed and does not change with the running of statements in the stored procedure.
state_change	timestamp with time zone	Time when **state** was last modified.

Name	Type	Description
waiting	Boolean	Whether the backend is currently waiting for a lock. If yes, the value is **true**.
enqueue	text	Unsupported currently.
state	text	Overall status of the backend. The value must be one of the following: • **active**: The backend is executing a query. • **idle**: The backend is waiting for a new client command. • **idle in transaction**: The backend is in a transaction, but there is no statement being executed in the transaction. • **idle in transaction** (aborted): The backend is in a transaction, but there are statements failed in the transaction. • **fastpath function call**: The backend is executing a fast-path function. • **disabled**: This state is reported if **track_activities** is disabled in this backend. **NOTE** Common users can view their own session status only. The state information of other accounts is empty. For example, after user **judy** is connected to the database, the state information of user **joe** and the initial user omm in **pg_stat_activity** is empty. `SELECT datname, usename, usesysid, state,pid FROM pg_stat_activity;` ` datname  | usename | usesysid | state  | pid` `----------+---------+----------+--------+-----------------` ` testdb | omm     |       10 |        | 139968752121616` ` testdb | omm     |       10 |        | 139968903116560` ` db_tpcc | judy    |    16398 | active | 139968391403280` ` testdb | omm     |       10 |        | 139968643069712` ` testdb | omm     |       10 |        | 139968680818448` ` testdb | joe     |    16390 |        | 139968563377936` `(6 rows)`
resource_pool	name	Resource pool used by the user.

Name	Type	Description
query_id	bigint	ID of a query.
query	text	Latest query at the backend. If the value of **state** is **active**, this column shows the ongoing query. In all other states, it shows the last query that was executed.
connection_info	text	A string in JSON format recording the driver type, driver version, driver deployment path, and process owner of the connected database. For details, see the GUC parameter **connection_info**.
unique_sql_id	bigint	Unique SQL statement ID.
trace_id	text	Driver-specific trace ID, which is associated with an application request.
top_xid	xid	Top-level transaction ID of a transaction.
current_xid	xid	Current transaction ID of a transaction.
xlog_quantity	bigint	Amount of Xlogs currently used by a transaction, in bytes.

12.3.12.160 PG_STAT_ACTIVITY_NG

PG_STAT_ACTIVITY_NG displays information about all queries in the logical database instance of the current user.

Table 12-384 PG_STAT_ACTIVITY_NG columns

Name	Type	Description
datid	oid	OID of the database that the user session connects to in the backend.
datname	name	Name of the database that the user session connects to in the backend.
pid	bigint	ID of the backend thread.
sessionid	bigint	Session ID.
usesysid	oid	OID of the user logged in to the backend.

Name	Type	Description
usename	name	Name of the user logged in to the backend.
application_name	text	Name of the application connected to the backend.
client_addr	inet	IP address of the client connected to the backend. If this column is **NULL**, it indicates either the client is connected via a Unix socket on the server or this is an internal thread, such as **AUTOVACUUM**.
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of **client_addr**. This column will be non-null only for IP connections and only when **log_hostname** is enabled.
client_port	integer	TCP port number that the client uses for communication with the backend (**–1** if a Unix socket is used).
backend_start	timestamp with time zone	Time when this session was started, that is, when the client connected to the server.
xact_start	timestamp with time zone	Time when the current active transaction was started (**NULL** if no transaction is active). If the current query is the first of its transaction, the value of this column is the same as that of the **query_start** column.
query_start	timestamp with time zone	Time when the current active query was started, or time when the last query was started if the value of **state** is not **active**. For a stored procedure, function, or package, the first query time is displayed and does not change with the running of statements in the stored procedure.
state_change	timestamp with time zone	Time when **state** was last modified.
waiting	Boolean	Specifies whether the backend is currently waiting for a lock. If yes, the value is **true**. Otherwise, the value is **false**.

Name	Type	Description
enqueue	text	Queuing status of a statement. The value can be: • **waiting in queue**: The statement is in the queue. • **Empty**: The statement is running.
state	text	Overall status of the backend. The value can be: • **active**: The backend is executing a query. • **idle**: The backend is waiting for a new client command. • **idle in transaction**: The backend is in a transaction, but there is no statement being executed in the transaction. • **idle in transaction** (aborted): The backend is in a transaction, but there are statements failed in the transaction. • **fastpath function call**: The backend is executing a fast-path function. • **disabled**: This state is reported if **track_activities** is disabled in this backend. **NOTE** Common users can view their own session status only. The state information of other accounts is empty. For example, after user **judy** is connected to the database, the state information of user **joe** and the initial user **omm** in **pg_stat_activity** is empty. SELECT datname, usename, usesysid, state,pid FROM pg_stat_activity_ng; datname | usename | usesysid | state | pid ----------+---------+----------+--------+----------------- testdb | omm | 10 | | 139968752121616 testdb | omm | 10 | | 139968903116560 db_tpcds | judy | 16398 | active | 139968391403280 testdb | omm | 10 | | 139968643069712 testdb | omm | 10 | | 139968680818448 testdb | joe | 16390 | | 139968563377936 (6 rows)

Name	Type	Description
resource_pool	name	Resource pool used by the user.
query_id	bigint	ID of a query.
query	text	Latest query at the backend. If the value of **state** is **active**, this column shows the ongoing query. In all other states, it shows the last query that was executed.
node_group	text	Logical database instance of the user to which the statement belongs.

12.3.12.161 PG_STAT_ALL_INDEXES

PG_STAT_ALL_INDEXES displays statistics about accesses to a specific index by querying each index row in the current database.

Indexes can be used via either simple index scans or bitmap index scans. In a bitmap scan the output of several indexes can be combined via AND or OR rules, so it is difficult to associate individual heap row fetches with specific indexes when a bitmap scan is used. Therefore, each bitmap scan increments the **pg_stat_all_indexes.idx_tup_read** count(s) for the index(es) it uses, and it increments the **pg_stat_all_tables.idx_tup_fetch** count for the table, but it does not affect **pg_stat_all_indexes.idx_tup_fetch**.

Table 12-385 PG_STAT_ALL_INDEXES columns

Name	Type	Description
relid	oid	OID of the table that the index is created for.
indexrelid	oid	OID of the index.
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table for the index
indexrelname	name	Index name
idx_scan	bigint	Number of index scans initiated on the index
idx_tup_read	bigint	Number of index entries returned by scans on the index.
idx_tup_fetch	bigint	Number of live rows fetched in the original table by a simple index scan that uses the index.

12.3.12.162 PG_STAT_ALL_TABLES

PG_STAT_ALL_TABLES queries one row for each table in the current database (including TOAST tables), showing statistics about a specific table.

Table 12-386 PG_STAT_ALL_TABLES columns

Name	Type	Description
relid	oid	OID of the table.
schemaname	name	Name of the schema that the table is in.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on the table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of rows HOT updated (that is, with no separate index update required).
n_live_tup	bigint	Estimated number of active rows.
n_dead_tup	bigint	Estimated number of inactive rows.
last_vacuum	timestamp with time zone	Last time when the table was cleared.
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Last time when the table was analyzed.
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.

Name	Type	Description
vacuum_count	bigint	Number of times the table is cleared.
autovacuum_count	bigint	Number of times that the table has been vacuumed by the autovacuum daemon.
analyze_count	bigint	Number of times the table is analyzed.
autoanalyze_count	bigint	Number of times that the table has been analyzed by the autovacuum daemon thread.
last_data_changed	timestamp with time zone	Last time when the data in the table changes (modifications that cause data changes on tables (insert/update/delete/truncate) and partitioned or subpartitioned tables (exchange/truncate/drop)). Data in this column is recorded only in the local database primary node.

12.3.12.163 PG_STAT_BAD_BLOCK

PG_STAT_BAD_BLOCK displays statistics about page verification failures after a node is started.

Table 12-387 PG_STAT_BAD_BLOCK columns

Name	Type	Description
nodename	text	Node name
databaseid	integer	OID of a database
tablespaceid	integer	Tablespace OID
relfilenode	integer	File object ID
bucketid	smallint	ID of the bucket for consistent hashing
forknum	integer	File type The values are as follows: **0**: main data file. **1**: FSM file. **2**: VM file. **3**: BCM file.
error_count	integer	Number of verification failures

Name	Type	Description
first_time	timestamp with time zone	Time of the first verification failure
last_time	timestamp with time zone	Time of the latest verification failure.

12.3.12.164 PG_STAT_BGWRITER

PG_STAT_BGWRITER displays statistics about the activity of the background writer process.

Table 12-388 PG_STAT_BGWRITER columns

Name	Type	Description
checkpoints_timed	bigint	Number of scheduled checkpoints that have been performed.
checkpoints_req	bigint	Number of checkpoints that have been performed actively.
checkpoint_write_time	double precision	Total time spent in the checkpoint processing portion when writing files to disk, in milliseconds.
checkpoint_sync_time	double precision	Total time spent in the checkpoint processing portion when synchronizing files to disk, in milliseconds.
buffers_checkpoint	bigint	Number of buffers written by checkpoints.
buffers_clean	bigint	Number of buffers written by the background writer process.
maxwritten_clean	bigint	Number of times that cleanup scanning stops because the background writer process writes too many buffers.
buffers_backend	bigint	Number of buffers written directly by the backend.

Name	Type	Description
buffers_backend_fsync	bigint	Number of times that the backend calls fsync (usually, even if the backend executes these write actions, the background writer process processes them again).
buffers_alloc	bigint	Number of buffers allocated.
stats_reset	timestamp with time zone	Time at which these statistics were last reset.

12.3.12.165 PG_STAT_DATABASE

PG_STAT_DATABASE contains database statistics for each database in GaussDB.

Table 12-389 PG_STAT_DATABASE columns

Name	Type	Description
datid	oid	OID of a database
datname	name	Name of the database
numbackends	integer	Number of backends currently connected to this database. This is the only column in this view that returns a value reflecting the current state; all other columns return the accumulated values since the last reset.
xact_commit	bigint	Number of transactions in this database that have been committed
xact_rollback	bigint	Number of transactions in this database that have been rolled back
blks_read	bigint	Number of disk blocks read in this database
blks_hit	bigint	Number of times disk blocks were found in the buffer cache (unnecessary as the number includes only hits in the database buffer cache)
tup_returned	bigint	Number of rows returned by queries in this database
tup_fetched	bigint	Number of rows fetched by queries in this database

Name	Type	Description
tup_inserted	bigint	Number of rows inserted by queries in this database
tup_updated	bigint	Number of rows updated by queries in this database
tup_deleted	bigint	Number of rows deleted by queries in this database
conflicts	bigint	Number of queries canceled due to database recovery conflicts (conflicts occurring only on the standby server). For details, see PG_STAT_DATABASE_CONFLICTS.
temp_files	bigint	Number of temporary files created by queries in this database. All temporary files are counted, regardless of why the temporary file was created (for example, sorting or hashing), and regardless of the **log_temp_files** setting.
temp_bytes	bigint	Total amount of data written to temporary files by queries in this database. All temporary files are counted, regardless of why the temporary file was created, and regardless of the **log_temp_files** setting.
deadlocks	bigint	Number of deadlocks detected in this database
blk_read_time	double precision	Time spent reading data file blocks by backends in this database (unit: ms)
blk_write_time	double precision	Time spent writing into data file blocks by backends in this database (unit: ms)
stats_reset	timestamp with time zone	Time at which the current statistics were reset

12.3.12.166 PG_STAT_DATABASE_CONFLICTS

PG_STAT_DATABASE_CONFLICTS displays statistics about database conflicts.

Table 12-390 PG_STAT_DATABASE_CONFLICTS columns

Name	Type	Description
datid	oid	Database ID
datname	name	Database name

Name	Type	Description
confl_tablespace	bigint	Number of conflicting tablespaces
confl_lock	bigint	Number of conflicting locks
confl_snapshot	bigint	Number conflicting snapshots
confl_bufferpin	bigint	Number of conflicting buffers
confl_deadlock	bigint	Number of conflicting deadlocks

12.3.12.167 PG_STAT_REPLICATION

PG_STAT_REPLICATION displays information about the log synchronization thread, such as the locations where the sender sends logs and where the receiver receives logs.

Table 12-391 PG_STAT_REPLICATION columns

Name	Type	Description
pid	bigint	Process ID of the thread
usesysid	oid	User system ID
usename	name	Username
application_name	text	Program name
client_addr	inet	Client address
client_hostname	text	Client name
client_port	integer	Port of the client
backend_start	timestamp with time zone	Start time of the program

Name	Type	Description
state	text	State of the log synchronization thread: • **startup**: The thread is being started. • **catchup**: The thread is establishing a connection between the standby server and the primary server. • **streaming**: The thread has established a connection between the standby server and the primary server and is replicating data streams. • **backup**: The thread is sending a backup. • **stopping**: The thread is being stopped.
sender_sent_location	text	Location where the sender sends logs
receiver_write_location	text	Location where the receiver writes logs
receiver_flush_location	text	Location where the receive end flushes logs
receiver_replay_location	text	Location where the receive end replays logs
sync_priority	integer	Priority of synchronous duplication (**0** indicates asynchronization.)

Name	Type	Description
sync_state	text	Synchronization state • **async**: asynchronous replication. • **sync**: synchronous replication. • **potential**: The standby server is asynchronous currently, but if a current synchronization server fails, the standby server becomes synchronous. • Qurom: switches between the synchronous and asynchronous states to ensure that there are more than a certain number of synchronous standby servers. Generally, the number of synchronous standby servers is (n+1)/2-1, where *n* indicates the total number of copies. Whether the standby server is synchronous depends on whether logs are received first. For details, see the description of the **synchronous_standby_names** parameter.

12.3.12.168 PG_STAT_SYS_INDEXES

PG_STAT_SYS_INDEXES displays index status information about all the system catalogs in the pg_catalog and information_schema schemas.

Table 12-392 PG_STAT_SYS_INDEXES columns

Name	Type	Description
relid	oid	OID of the table that the index is created for.
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table for the index
indexrelname	name	Index name
idx_scan	bigint	Number of index scans initiated on the index
idx_tup_read	bigint	Number of index entries returned by scans on the index.
idx_tup_fetch	bigint	Number of live rows fetched in the original table by a simple index scan that uses the index.

12.3.12.169 PG_STAT_SYS_TABLES

PG_STAT_SYS_TABLES displays statistics about the system catalogs of all the namespaces in pg_catalog and information_schema schemas.

Table 12-393 PG_STAT_SYS_TABLES columns

Name	Type	Description
relid	oid	OID of the table.
schemaname	name	Name of the schema that the table is in.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on the table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.

Name	Type	Description
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required).
n_live_tup	bigint	Estimated number of live rows.
n_dead_tup	bigint	Estimated number of inactive rows.
last_vacuum	timestamp with time zone	Last time when this table was manually vacuumed (excluding **VACUUM FULL**)
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Last time when the table was manually analyzed.
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.
vacuum_count	bigint	Number of times that the table has been manually vacuumed (not counting **VACUUM FULL**).
autovacuum_count	bigint	Number of times the table has been vacuumed by the autovacuum daemon thread.
analyze_count	bigint	Number of times that the table has been manually analyzed.
autoanalyze_count	bigint	Number of times that the table has been analyzed by the autovacuum daemon thread.
last_data_changed	timestamp with time zone	Last modification time of the table data.

12.3.12.170 PG_STAT_USER_FUNCTIONS

PG_STAT_USER_FUNCTIONS shows user-defined function status information in the namespace. (The language of the function is non-internal language.)

Table 12-394 PG_STAT_USER_FUNCTIONS columns

Name	Type	Description
funcid	oid	OID of a function

Name	Type	Description
schemaname	name	Schema name
funcname	name	Function name
calls	bigint	Number of times the function has been called
total_time	double precision	Total time spent in the function and all other functions called by it
self_time	double precision	Total time spent in the function itself, excluding other functions called by it

12.3.12.171 PG_STAT_USER_INDEXES

PG_STAT_USER_INDEXES displays information about the index status of user-defined ordinary tables and TOAST tables.

Table 12-395 PG_STAT_USER_INDEXES columns

Name	Type	Description
relid	oid	OID of the table that the index is created for.
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table for the index
indexrelname	name	Index name
idx_scan	bigint	Number of index scans initiated on the index
idx_tup_read	bigint	Number of index entries returned by scans on the index.
idx_tup_fetch	bigint	Number of live rows fetched in the original table by a simple index scan that uses the index.

12.3.12.172 PG_STAT_USER_TABLES

PG_STAT_USER_TABLES displays information about user-defined ordinary tables and TOAST tables in the namespaces.

Table 12-396 PG_STAT_USER_TABLES columns

Name	Type	Description
relid	oid	OID of the table.
schemaname	name	Name of the schema that the table is in.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on the table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required).
n_live_tup	bigint	Estimated number of live rows.
n_dead_tup	bigint	Estimated number of inactive rows.
last_vacuum	timestamp with time zone	Last time when this table was manually vacuumed (excluding **VACUUM FULL**)
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Last time when the table was manually analyzed.
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.
vacuum_count	bigint	Number of times that the table has been manually vacuumed (not counting **VACUUM FULL**).

Name	Type	Description
autovacuum_count	bigint	Number of times the table has been vacuumed by the autovacuum daemon thread.
analyze_count	bigint	Number of times that the table has been manually analyzed.
autoanalyze_count	bigint	Number of times that the table has been analyzed by the autovacuum daemon thread.
last_data_changed	timestamp with time zone	Last modification time of the table data.

12.3.12.173 PG_STAT_XACT_ALL_TABLES

PG_STAT_XACT_ALL_TABLES displays transaction status information about all ordinary tables and TOAST tables in the namespaces.

Table 12-397 PG_STAT_XACT_ALL_TABLES columns

Name	Type	Description
relid	oid	OID of the table
schemaname	name	Name of the schema that the table is in
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)

12.3.12.174 PG_STAT_XACT_SYS_TABLES

PG_STAT_XACT_SYS_TABLES displays transaction status information of the system catalog in the namespace.

Table 12-398 PG_STAT_XACT_SYS_TABLES columns

Name	Type	Description
relid	oid	OID of the table
schemaname	name	Name of the schema that the table is in
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)

12.3.12.175 PG_STAT_XACT_USER_FUNCTIONS

PG_STAT_XACT_USER_FUNCTIONS contains statistics on the execution of each function.

Table 12-399 PG_STAT_XACT_USER_FUNCTIONS columns

Name	Type	Description
funcid	oid	OID of a function
schemaname	name	Schema name
funcname	name	Function name
calls	bigint	Number of times the function has been called
total_time	double precision	Total time spent in the function and all other functions called by it
self_time	double precision	Total time spent in the function itself, excluding other functions called by it

12.3.12.176 PG_STAT_XACT_USER_TABLES

PG_STAT_XACT_USER_TABLES displays transaction status information of the user table in the namespace.

Table 12-400 PG_STAT_XACT_USER_TABLES columns

Name	Type	Description
relid	oid	OID of the table
schemaname	name	Name of the schema that the table is in
relname	name	Table name
seq_scan	bigint	Number of sequential scans initiated on the table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of rows HOT updated (with no separate index update required)

12.3.12.177 PG_STATIO_ALL_INDEXES

PG_STATIO_ALL_INDEXES contains one row for each index in the current database, showing I/O statistics about accesses to that specific index.

Table 12-401 PG_STATIO_ALL_INDEXES columns

Name	Type	Description
relid	oid	OID of the table for this index
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	bigint	Number of disk blocks read from the index

Name	Type	Description
idx_blks_hit	bigint	Number of cache hits in the index

12.3.12.178 PG_STATIO_ALL_SEQUENCES

PG_STATIO_ALL_SEQUENCES contains the I/O statistics of each sequence in the current database.

Table 12-402 PG_STATIO_ALL_SEQUENCES columns

Name	Type	Description
relid	oid	OID of this sequence
schemaname	name	Name of the schema where the sequence is in
relname	name	Name of the sequence
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Cache hits in the sequence

12.3.12.179 PG_STATIO_ALL_TABLES

PG_STATIO_ALL_TABLES contains I/O statistics for each table (including the TOAST table) in the current database.

Table 12-403 PG_STATIO_ALL_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Name of the schema that the table is in
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from the table
heap_blks_hit	bigint	Number of cache hits in the table
idx_blks_read	bigint	Number of disk blocks read from all indexes in the table
idx_blks_hit	bigint	Number of cache hits in indexes in the table
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in the table

Name	Type	Description
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in the table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in the table
tidx_blks_hit	bigint	Number of buffer-hits in the TOAST table index (if any) in the table

12.3.12.180 PG_STATIO_SYS_INDEXES

PG_STATIO_SYS_INDEXES displays I/O status information for all system catalog indexes in a namespace.

Table 12-404 PG_STATIO_SYS_INDEXES columns

Name	Type	Description
relid	oid	OID of the table for this index
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	bigint	Number of disk blocks read from the index
idx_blks_hit	bigint	Number of cache hits in the index

12.3.12.181 PG_STATIO_SYS_SEQUENCES

PG_STATIO_SYS_SEQUENCES displays I/O status information about all the sequences in the namespace.

Table 12-405 PG_STATIO_SYS_SEQUENCES columns

Name	Type	Description
relid	oid	OID of this sequence
schemaname	name	Name of the schema where the sequence is in
relname	name	Name of the sequence

Name	Type	Description
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Cache hits in the sequence

12.3.12.182 PG_STATIO_SYS_TABLES

PG_STATIO_SYS_TABLES shows I/O status information about all the system catalogs in the namespace.

Table 12-406 PG_STATIO_SYS_TABLES columns

Name	Type	Description
relid	oid	Table OID
schemaname	name	Name of the schema the table is in
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from the table
heap_blks_hit	bigint	Number of cache hits in the table
idx_blks_read	bigint	Number of disk blocks read from all indexes in the table
idx_blks_hit	bigint	Number of cache hits in indexes in the table
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in the table
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in the table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in the table
tidx_blks_hit	bigint	Number of buffer-hits in the TOAST table index (if any) in the table

12.3.12.183 PG_STATIO_USER_INDEXES

PG_STATIO_USER_INDEXES displays I/O status information about all the user relationship table indexes in the namespace.

Table 12-407 PG_STATIO_USER_INDEXES columns

Name	Type	Description
relid	oid	OID of the table that the index is created for
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table for the index
indexrelname	name	Index name
idx_blks_read	bigint	Number of disk blocks read from the index
idx_blks_hit	bigint	Number of cache hits in the index

12.3.12.184 PG_STATIO_USER_SEQUENCES

PG_STATIO_USER_SEQUENCES shows I/O status information about all the user relationship table sequences in the namespace.

Table 12-408 PG_STATIO_USER_SEQUENCES columns

Name	Type	Description
relid	oid	OID of this sequence
schemaname	name	Name of the schema where the sequence is in
relname	name	Name of the sequence
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Cache hits in the sequence

12.3.12.185 PG_STATIO_USER_TABLES

PG_STATIO_USER_TABLES displays I/O status information about all the user relationship tables in the namespace.

Table 12-409 PG_STATIO_USER_TABLES columns

Name	Type	Description
relid	oid	Table OID

Name	Type	Description
schemaname	name	Name of the schema that the table is in
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from the table
heap_blks_hit	bigint	Number of cache hits in the table
idx_blks_read	bigint	Number of disk blocks read from all indexes in the table
idx_blks_hit	bigint	Number of cache hits in indexes in the table
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in the table
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in the table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in the table
tidx_blks_hit	bigint	Number of buffer-hits in the TOAST table index (if any) in the table

12.3.12.186 PG_TABLES

PG_TABLES provides access to each table in the database.

Table 12-410 PG_TABLES columns

Name	Type	Reference	Description
schemaname	name	PG_NAMESPACE.nspname	Name of the schema that contains a table
tablename	name	PG_CLASS.relname	Table name
tableowner	name	pg_get_userbyid(PG_CLASS.relowner)	Table owner
tablespace	name	PG_TABLESPACE.spcname	Tablespace that contains the table. The default value is null.
hasindexes	boolean	PG_CLASS.relhasindex	The value is **TRUE** if the table has (or recently had) an index; otherwise, the value is **FALSE**.
hasrules	boolean	PG_CLASS.relhasruls	The value is **TRUE** if the table has rules; otherwise, the value is **FALSE**.

Name	Type	Reference	Description
hastriggers	boolean	PG_CLASS.RELHASTRIGGERS	The value is **TRUE** if the table has triggers; otherwise, the value is **FALSE**.
tablecreator	name	pg_get_userbyid(po.creator)	Table creator.
created	timestamp with time zone	pg_object.ctime	Creation time of the object
last_ddl_time	timestamp with time zone	pg_object.mtime	Last modification time of the object

12.3.12.187 PG_THREAD_WAIT_STATUS

PG_THREAD_WAIT_STATUS allows you to test the block waiting status about the backend thread and auxiliary thread of the current instance.

Table 12-411 PG_THREAD_WAIT_STATUS columns

Name	Type	Description
node_name	text	Current node name.
db_name	text	Database name.
thread_name	text	Thread name.
query_id	bigint	Query ID. It is equivalent to **debug_query_id**.
tid	bigint	Thread ID of the current thread.
sessionid	bigint	Current session ID.
lwtid	integer	Lightweight thread ID of the current thread.
psessionid	bigint	Parent session ID.
tlevel	integer	Level of the streaming thread.
smpid	integer	Concurrent thread ID.
wait_status	text	Waiting status of the current thread. For details about the waiting status, see **Table 12-412**.

Name	Type	Description
wait_event	text	If **wait_status** is set to **acquire lock**, **acquire lwlock**, or **wait io**, this column describes the lock, lightweight lock, and I/O information, respectively. Otherwise, this column is empty.
locktag	text	Information about the lock that the current thread is waiting to obtain.
lockmode	text	Lock mode that the current thread is waiting to obtain. The values include table-level lock, row-level lock, and page-level lock modes.
block_sessionid	bigint	ID of the session that blocks the current thread from obtaining the lock.
global_sessionid	text	Global session ID.

The waiting states in the **wait_status** column are as follows.

Table 12-412 Waiting states

Value	Description
none	Waiting for no event.
acquire lock	Waiting for locking until the locking succeeds or times out.
acquire lwlock	Waiting for a lightweight lock.
wait io	Waiting for I/O completion.
wait cmd	Waiting for reading network communication packets to complete.
wait pooler get conn	Waiting for pooler to obtain connections.
wait pooler abort conn	Waiting for pooler to terminate connections.
wait pooler clean conn	Waiting for pooler to clear connections.
pooler create conn: [nodename], total N	Waiting for the pooler to set up a connection. The connection is being established with the node specified by *nodename*, and there are *N* connections waiting to be set up.
get conn	Obtaining the connection to other nodes.

Value	Description
set cmd: [nodename]	Waiting for running the **SET**, **RESET**, **TRANSACTION BLOCK LEVEL PARA SET**, or **SESSION LEVEL PARA SET** statement on the connection. The statement is being executed on the node specified by *nodename*.
cancel query	Canceling the SQL statement that is being executed through the connection.
stop query	Stopping the query that is being executed through the connection.
wait node: [nodename](plevel), total N, [phase]	Waiting for receiving data from a connected node. The thread is waiting for data from the plevel thread of the node specified by *nodename*. The data of *N* connections is waiting to be returned. If *phase* is included, the possible phases are as follows: • **begin**: The transaction is being started. • **commit**: The transaction is being committed. • **rollback**: The transaction is being rolled back.
wait transaction sync: xid	Waiting for synchronizing the transaction specified by *xid*.
wait wal sync	Waiting for the completion of WAL of synchronization from the specified LSN to the standby instance.
wait data sync	Waiting for the completion of data page synchronization to the standby instance.
wait data sync queue	Waiting for putting the data pages that are in the row-store into the synchronization queue.
flush data: [nodename](plevel), [phase]	Waiting for sending data to the plevel thread of the node specified by *nodename*. If *phase* is included, the possible phase is **wait quota**, indicating that the current communication flow is waiting for the quota value.
stream get conn: [nodename], total N	Waiting for connecting to the consumer object of the node specified by *nodename* when the stream flow is initialized. There are *N* consumers waiting to be connected.

Value	Description
wait producer ready: [nodename] (plevel), total N	Waiting for each producer to be ready when the stream flow is initialized. The thread is waiting for the procedure of the plevel thread on the *nodename* node to be ready. There are *N* producers waiting to be ready.
synchronize quit	Waiting for the threads in the stream thread group to quit when the stream plan ends.
wait stream nodegroup destroy	Waiting for destroying the stream node group when the stream plan ends.
wait active statement	Waiting for job execution under resource and load control.
analyze: [relname], [phase]	Executing ANALYZE for the *relname* table. If *phase* is included, the possible phase is **autovacuum**, indicating that the database automatically enables the AutoVacuum thread to execute **ANALYZE**.
vacuum: [relname], [phase]	Executing VACUUM for the *relname* table. If *phase* is included, the possible phase is **autovacuum**, indicating that the database automatically enables the AutoVacuum thread to execute **VACUUM**.
vacuum full: [relname]	Executing VACUUM FULL for the *relname* table.
create index	Creating an index.
HashJoin - [build hash | write file]	The **HashJoin** operator is being executed. In this phase, you need to pay attention to the execution time-consuming. • **build hash**: The **HashJoin** operator is creating a hash table. • **write file**: The **HashJoin** operator is writing data to disks.
HashAgg - [build hash | write file]	The **HashAgg** operator is being executed. In this phase, you need to pay attention to the execution time-consuming. • **build hash**: The **HashAgg** operator is creating a hash table. • **write file**: The **HashAgg** operator is writing data to disks.

Value	Description
HashSetop - [build hash | write file]	The **HashSetop** operator is being executed. In this phase, you need to pay attention to the execution time-consuming. • **build hash**: The **HashSetop** operator is creating a hash table. • **write file**: The **HashSetop** operator is writing data to disks.
Sort | Sort - [fetch tuple | write file]	The **Sort** operator is used for sorting. **fetch tuple** indicates that the **Sort** operator is obtaining tuples, and **write file** indicates that the **Sort** operator is writing data to disks.
Material | Material - write file	The **Material** operator is being executed. **write file** indicates that the **Material** operator is writing data to disks.
NestLoop	The **NestLoop** operator is being executed.
wait memory	Waiting for obtaining the memory.
wait sync consumer next step	Waiting for the consumer to execute the stream operator.
wait sync producer next step	Waiting for the producer to execute the stream operator.
vacuum gpi	Clearing the global partitioned index in the vacuum or autovacuum process.
standby read recovery conflict	The read-only mode of the standby node conflicts with the log replay mode.
standby get snapshot	The standby node obtains the snapshot in read-only mode.
prune table	Waiting for a heap table to clear historical deleted data.
prune index	Waiting for an index to clear historical deleted data.
wait reserve td	Waiting for the Ustore transaction slot allocation.
wait td rollback	Waiting for the Ustore transaction slot to roll back transactions.
wait available td	Waiting for an available transaction slot for Ustore.
wait transaction rollback	Waiting for a transaction to roll back.

Value	Description
wait sync bgworkers	Waiting for an index subthread created in parallel to complete local scanning and sorting.
wait io control	Waiting for job execution under I/O control.
wait gs_sleep	Waiting for the server thread delay.
wait vacuum delay	Waiting for VACUUM delay.
wait seq scan	Execution time of seq scan.
wait index scan	Execution time of index scan.
wait checkpoint start	Start time of the checkpoint phase.
wait checkpoint done	End time of the checkpoint phase.
wait xact start command	Start time of xact.
wait xact commit command	Commit time of xact.
wait for autoextend partition	Waiting for the partitioned table to automatically expand and create partitions.
Accept client conn - Total Time	Total duration from the time when a connection is established to the time when the connection is successful (after the GaussDB receives a connection request from a client).
Accept client conn - ThrdPool - add epoll	Duration from the time when a connection is established to the time when the thread pool adds the session handle to the epoll in thread pool mode (after the GaussDB receives a connection request from a client).
Accept client conn - ThrdPool - wait worker	Duration from the time when the thread pool adds the session handle to the epoll to the time when the working thread pool starts to work in thread pool mode (after the GaussDB receives a connection request from a client).
Accept client conn - ThrdPool - init session	Duration from the time when the working thread pool starts to work to the time the session initialization ends in thread pool mode (after the GaussDB receives a connection request from a client).
Accept client conn - Worker - init proc	(After the GaussDB receives a connection request from a client) Duration from the time when a connection is established to the time when the thread initialization ends in non-thread pool mode.

Value	Description
Accept client conn - Worker - init session	(After the GaussDB receives a connection request from a client) Duration from the time when the thread initialization ends to the time when the session initialization ends in non-thread pool mode.
security audit write pipe	Waiting for an audit log to be written to the pipe.

If **wait_status** is **acquire lwlock**, **acquire lock**, or **wait io**, there is an event performing I/O operations or waiting for obtaining the corresponding lightweight lock or transaction lock.

The following table describes the corresponding wait events when **wait_status** is **acquire lwlock**. If **wait_event** is **extension**, the lightweight lock is dynamically allocated and is not monitored.

Table 12-413 List of wait events corresponding to lightweight locks

wait_event	Description
ShmemIndexLock	Used to protect the primary index table, a hash table, in shared memory.
OidGenLock	Used to prevent different threads from generating the same OID.
XidGenLock	Used to prevent two transactions from obtaining the same XID.
ProcArrayLock	Used to prevent concurrent access to or concurrent modification on the ProcArray shared array.
SInvalReadLock	Used to prevent concurrent execution with invalid message deletion.
SInvalWriteLock	Used to prevent concurrent execution with invalid message write and deletion.
WALInsertLock	Used to prevent concurrent execution with WAL insertion.
WALWriteLock	Used to prevent concurrent write from a WAL buffer to a disk
ControlFileLock	Used to prevent concurrent read/write or concurrent write/write on the **pg_control** file.
CheckpointLock	Used to prevent multi-checkpoint concurrent execution.

wait_event	Description
CLogControlLock	Used to prevent concurrent access to or concurrent modification on the Clog control data structure.
SubtransControlLock	Used to prevent concurrent access to or concurrent modification on the subtransaction control data structure.
MultiXactGenLock	Used to allocate a unique MultiXact ID in serial mode.
MultiXactOffsetControlLock	Used to prevent concurrent read/write or concurrent write/write on **pg_multixact/offset**.
MultiXactMemberControlLock	Used to prevent concurrent read/write or concurrent write/write on **pg_multixact/members**.
RelCacheInitLock	Used to add a lock before any operations are performed on the **init** file when messages are invalid.
CheckpointerCommLock	Used to send file flush requests to a checkpointer. The request structure needs to be inserted to a request queue in serial mode.
TwoPhaseStateLock	Used to prevent concurrent access to or modification on two-phase information sharing arrays.
TablespaceCreateLock	Used to check whether a tablespace exists.
BtreeVacuumLock	Used to prevent VACUUM from clearing pages that are being used by B-tree indexes.
AutovacuumLock	Used to access the autovacuum worker array in serial mode.
AutovacuumScheduleLock	Used to distribute tables requiring VACUUM in serial mode.
AutoanalyzeLock	Used to obtain and release resources related to a task that allows for autoanalyze execution.
SyncScanLock	Used to determine the start position of a relfilenode during heap scanning.
NodeTableLock	Used to protect a shared structure that stores database node information.
PoolerLock	Used to prevent two threads from simultaneously obtaining the same connection from a connection pool.

wait_event	Description
RelationMappingLock	Used to wait for the mapping file between system catalogs and storage locations to be updated.
Async Ctl	Used to protect asynchronization buffers.
AsyncCtlLock	Used to prevent concurrent access to or concurrent modification on the sharing notification status.
AsyncQueueLock	Used to prevent concurrent access to or concurrent modification on the sharing notification queue.
SerializableXactHashLock	Used to prevent concurrent read/write or concurrent write/write on a sharing structure for serializable transactions.
SerializableFinishedListLock	Used to prevent concurrent read/write or concurrent write/write on a shared linked list for completed serial transactions.
SerializablePredicateLockList-Lock	Used to protect a linked list of serializable transactions that have locks.
OldSerXidLock	Used to protect a structure that records serializable transactions that have conflicts.
FileStatLock	Used to protect a data structure that stores statistics file information.
SyncRepLock	Used to protect Xlog synchronization information during primary-standby replication.
DataSyncRepLock	Used to protect data page synchronization information during primary-standby replication.
MetaCacheSweepLock	Used to add a lock when metadata is cyclically washed out.
ExtensionConnectorLibLock	Used to add a lock when a specific dynamic library is loaded or uninstalled in ODBC connection initialization scenarios.
SearchServerLibLock	Used to add a lock on the file read operation when a specific dynamic library is initially loaded in GPU-accelerated scenarios.
LsnXlogChkFileLock	Used to serially update the Xlog flush points for primary and standby servers recorded in a specific structure.

wait_event	Description
ReplicationSlotAllocationLock	Used to add a lock when a primary server allocates streaming replication slots during primary-standby replication.
ReplicationSlotControlLock	Used to prevent concurrent update of replication slot status during primary-standby replication.
ResourcePoolHashLock	Used to prevent concurrent access to or concurrent modification on a resource pool table, a hash table.
WorkloadStatHashLock	Used to prevent concurrent access to or concurrent modification on a hash table that contains SQL requests from the primary node of the database.
WorkloadIoStatHashLock	Used to prevent concurrent access to or concurrent modification on a hash table that contains I/O information of the current database node.
WorkloadCGroupHashLock	Used to prevent concurrent access to or concurrent modification on a hash table that contains Cgroup information.
OBSGetPathLock	Used to prevent concurrent read/write or concurrent write/write on an OBS path.
WorkloadUserInfoLock	Used to prevent concurrent access to or concurrent modification on a hash table that contains user information about load management.
WorkloadRecordLock	Used to prevent concurrent access to or concurrent modification on a hash table that contains requests received by the primary node of the databases during adaptive memory management.
WorkloadIOUtilLock	Used to protect a structure that records **iostat** and CPU load information.
WorkloadNodeGroupLock	Used to prevent concurrent access to or concurrent modification on a hash table that contains node group information in memory.
JobShmemLock	Used to protect global variables in the shared memory that is periodically read during a scheduled job.
OBSRuntimeLock	Used to obtain environment variables, for example, **GASSHOME**.

wait_event	Description
LLVMDumpIRLock	Used to export the assembly language for dynamically generating functions.
LLVMParseIRLock	Used to compile and parse a finished IR function from the IR file at the start position of a query.
CriticalCacheBuildLock	Used to load caches from a shared or local cache initialization file.
WaitCountHashLock	Used to protect a shared structure in user statement counting scenarios.
BufMappingLock	Used to protect operations on a table mapped to shared buffer.
LockMgrLock	Used to protect a common lock structure.
PredicateLockMgrLock	Used to protect a lock structure that has serializable transactions.
OperatorRealTLock	Used to prevent concurrent access to or concurrent modification on a global structure that contains real-time data at the operator level.
OperatorHistLock	Used to prevent concurrent access to or concurrent modification on a global structure that contains historical data at the operator level.
SessionRealTLock	Used to prevent concurrent access to or concurrent modification on a global structure that contains real-time data at the query level.
SessionHistLock	Used to prevent concurrent access to or concurrent modification on a global structure that contains historical data at the query level.
BarrierLock	Used to ensure that only one thread is creating a barrier at a time
dummyServerInfoCacheLock	Used to protect a global hash table where the information about database connections is cached.
RPNumberLock	Used by a database node on a computing GaussDB to count the number of threads for a task where plans are being executed.
CBMParseXlogLock	Used to protect the lock used when CBM parses Xlogs.
RelfilenodeReuseLock	Used to prevent the link to a reused column attribute file from being canceled by mistake.

wait_event	Description
RcvWriteLock	Used to prevent concurrent call of **WalDataRcvWrite**.
PercentileLock	Used to protect global percentile buffers.
CSNBufMappingLock	Used to protect CSN pages.
UniqueSQLMappingLock	Used to protect a unique SQL hash table.
DelayDDLLock	Used to prevent concurrent DDL operations.
CLOG Ctl	Used to prevent concurrent access to or concurrent modification on the Clog control data structure.
Async Ctl	Used to protect asynchronization buffers.
MultiXactOffset Ctl	Used to protect SLRU buffers of a MultiXact offset.
MultiXactMember Ctl	Used to protect SLRU buffers of a MultiXact member.
OldSerXid SLRU Ctl	Used to protect SLRU buffers of old transaction IDs.
ReplicationSlotLock	Used to protect a replication slot.
PGPROCLock	Used to protect the PGPROC structure.
MetaCacheLock	Used to protect meta caches.
DataCacheLock	Used to protect data caches.
InstrUserLock	Used to protect InstrUserHTAB.
BadBlockStatHashLock	Used to protect the hash table **global_bad_block_stat**.
BufFreelistLock	Used to ensure the atomicity of free list operations in the shared buffer.
AddinShmemInitLock	Used to protect the initialization of the shared memory object.
AlterPortLock	Used to protect the coordinator node from changing the registration port number.
FdwPartitionCaheLock	Management lock of the buffer of the HDFS partitioned table.
DfsConnectorCacheLock	Management lock of the DFSConnector buffer.
DfsSpaceCacheLock	Management lock of the HDFS tablespace management buffer.

wait_event	Description
FullBuildXlogCopyStartPtrLock	Used to protect Xlog copy operations in the full build.
DfsUserLoginLock	Used for HDFS user login and authentication.
LogicalReplicationSlotPersis-tentDataLock	Used to protect data in the replication slot during logical replication.
PgfdwLock	Used by the management instance to establish a connection to the foreign server.
InstanceTimeLock	Used to obtain time information of sessions in an instance.
XlogRemoveSegLock	Used to protect Xlog segment file recycling.
DnUsedSpaceHashLock	Used to update space usage information corresponding to a session.
CsnMinLock	Used to calculate CSNmin.
GPCCommitLock	Used to protect the addition of the global Plan Cache hash table.
GPCClearLock	Used to protect the clearing of the global plan cache hash table.
GPCTimelineLock	Used to protect the timeline check of the global plan cache hash table.
InstanceRealTLock	Used to protect the update of the hash table that stores shared instance statistics.
CLogBufMappingLock	Used to manage the cache of commit logs.
GPCMappingLock	Used to manage the global plan cache.
GPCPrepareMappingLock	Used to manage the global plan cache.
BufferIOLock	Used to protect I/O operations on pages in the shared buffer.
BufferContentLock	Used to protect the read and modification of the page content in the shared buffer.
CSNLOG Ctl	Used for CSN log management.
DoubleWriteLock	Used to manage doublewrite operations.
RowPageReplicationLock	Used to manage data page replication of row-store.
MatviewSeqnoLock	Used to manage the cache of materialized views.

wait_event	Description
GPRCMappingLock	Used to manage the access and modification operations of the global cache hash table of autonomous transactions.
extension	Other lightweight locks.
wait active statement	Waiting for job execution under resource and load control.
wait memory	Waiting for obtaining the memory.
IOStatLock	Used to concurrently maintain the hash table of resource management I/O statistics.
StartBlockMappingLock	Used by globalstat to obtain information such as startblockarray from pgstat.
PldebugLock	Used to debug stored procedures and perform concurrent maintenance operations.
DataFileIdCacheLock	Used to manage the concurrent access and storage of hash table that stores data file descriptor in the shared memory.
GTMHostInfoLock	Used to protect the concurrent access and storage of shared GTM host information.
TwoPhaseStatePartLock	Used to protect the status information of two-phase transactions (in each partition).
WALBufMappingLock	Used to protect the mapping between WAL buffer page and LSN offset.
UndoZoneLock	Used to protect the concurrent access and storage of undo zone.
RollbackReqHashLock	Used to protect the concurrent access and storage of hash table that stores the rollback request in the shared memory.
UHeapStatLock	Used to protect the concurrent access and storage of Ustore statistics.
WALWritePaxosLock	Used to protect the concurrent sequence of WAL log written to the paxos replication component.
SyncPaxosLock	Used to protect the concurrent access and storage of paxos synchronization queue.
BackgroundWorkerLock	Used to protect the concurrent sequence of background workers.
HadrSwitchoverLock	Used to protect the concurrent sequence of DR switchover.

wait_event	Description
HashUidLock	Used to protect the concurrent sequence of UID allocation.
ParallelDecodeLock	Used to protect the concurrent sequence of parallel decoding.
XLogMaxCSNLock	Used to protect the maximum volume of CSNs that can be restored locally in DR mode.
DisasterCacheLock	Used to protect the concurrent access and storage of DR cache information in the shared memory.
MaxCSNArrayLock	Used to protect the restoration progress of standby node CSNs in each shard in the shared memory.
RepairBadBlockStatHashLock	Used to protect the concurrent access and storage of the hash table that stores damaged pages in the shared memory.
HypoIndexLock	Used to create, delete and reset the virtual index by the lightweight lock.
XGBoostLibLock	Used by DB4AI to call the XGBoost library.
DropArchiveSlotLock	Used to protect the concurrent sequence of deleting the archive slot.
ProcXactMappingLock	Used to protect the concurrent access and storage of the hash table that stores the mapping information between transaction IDs and threads.
UndoPerZoneLock	Used to protect the concurrent access and storage of each information in undo zone.
UndoSpaceLock	Used to protect the concurrent access and storage of undo space.
SnapshotBlockLock	Used to control the concurrent sequence of snapshot-based backup and disk flushing.
DWSingleFlushFirstLock	Used to control the concurrent sequence of non-segment-pages and single-page doublewrite files.
DWSingleFlushSecondLock	Used to control the concurrent sequence of segment-pages and single-page doublewrite files.
DWSingleFlushSecondBufTag-Lock	Used to control the concurrent access and storage of metadata of segment-pages and single-page doublewrite files.

wait_event	Description
RestartPointQueueLock	Used to control the concurrent access and storage of restart point arrays on the standby node.
UnlinkRelHashTblLock	Used to protect the concurrent access and storage of the hash table that restores files to be deleted.
UnlinkRelForkHashTblLock	Used to protect the concurrent access and storage of the hash table that restores the fork files to be deleted.
WALFlushWait	Used to protect the concurrent sequence of log flushing.
WALConsensusWait	Used to protect the transaction commit or log replay which is performed only when the logs are consistent.
WALBufferInitWait	Used to protect the initialization and disk flushing sequence of WAL pages in the shared memory.
WALInitSegment	Used to protect the initialization sequence of WAL segment files.
SegmentHeadPartitionLock	Used to protect the partition lock of metadata in the segment-page header.
PgwrSyncQueueLock	Used to protect the concurrent access and storage of file queues to be flushed to disks.
BarrierHashTblLock	Used to protect the concurrent access and storage of barrier tables in the shared memory.
PageRepairHashTblLock	Used to protect the concurrent access and storage of hash table that stores repaired pages.
FileRepairHashTblLock	Used to protect the concurrent access and storage of the hash table that stores repaired files.
InstrUserLockId	Used to protect the concurrent modification of the hash table that stores user login and logout records.
GsStackLock	Used to ensure that the gs_tack function is not concurrently invoked.
InstrStmtTrackCtlLock	Used to protect the concurrent access and storage of the hash table when full SQL statements are dynamically enabled.

wait_event	Description
CaptureViewFileHashLock	Used to protect the concurrent access and storage of the hash table when capturing the performance view.
UniqueSqlEvictLock	Used to protect the concurrent access and storage of the hash table when the unique SQL reclamation is enabled.
gtt_shared_ctl	Used to protect concurrent read and write of the hash table that stores the global temporary tables.
AuditIndexFileLock	Used to protect concurrent read and write of index files in the audit log.
TDEKeyCacheLock	Used to control the concurrent read and write of cached data keys encrypted transparently.
BlockchainVersionLock	Used to control the concurrent read and write of global block numbers in the ledger database.
GlobalPrevHashLock	Used to control the concurrent read and write of global verification hash tables in the ledger database.
LWTRANCHE_ACCOUNT_TABLE	Used to control the concurrent read and write of the hash table that stores the account locking status.

The following table describes the corresponding wait events when **wait_status** is **wait io**.

Table 12-414 List of wait events corresponding to I/Os

wait_event	Description
BufFileRead	Reads data from a temporary file to a specified buffer.
BufFileWrite	Writes the content of a specified buffer to a temporary file.
ControlFileRead	Reads the **pg_control** file, mainly during database startup, checkpoint execution, and primary/standby verification.
ControlFileSync	Flushes the **pg_control** file to a disk, mainly during database initialization.

wait_event	Description
ControlFileSyncUpdate	Flushes the **pg_control** file to a disk, mainly during database startup, checkpoint execution, and primary/standby verification.
ControlFileWrite	Writes the **pg_control** file, during database initialization.
ControlFileWriteUpdate	Updates the **pg_control** file, mainly during database startup, checkpoint execution, and primary/standby verification.
CopyFileRead	Reads a file during file copying.
CopyFileWrite	Writes a file during file copying.
DataFileExtend	Writes a file during file name extension.
DataFileFlush	Flushes a table data file to a disk.
DataFileImmediateSync	Flushes a table data file to a disk immediately.
DataFilePrefetch	Reads a table data file asynchronously.
DataFileRead	Reads a table data file synchronously.
DataFileSync	Flushes table data file modifications to a disk.
DataFileTruncate	Truncates a table data file.
DataFileWrite	Writes a table data file.
LockFileAddToDataDirRead	Reads the **postmaster.pid** file.
LockFileAddToDataDirSync	Flushes the **postmaster.pid** file to a disk.
LockFileAddToDataDirWrite	Writes PID information into the **postmaster.pid** file.
LockFileCreateRead	Reads the LockFile file **%s.lock**.
LockFileCreateSync	Flushes the LockFile file **%s.lock** to a disk.
LockFileCreateWRITE	Writes PID information into the LockFile file **%s.lock**.
RelationMapRead	Reads the mapping file between system catalogs and storage locations.
RelationMapSync	Flushes the mapping file between system catalogs and storage locations to a disk.
RelationMapWrite	Writes the mapping file between system catalogs and storage locations.
ReplicationSlotRead	Reads a streaming replication slot file during a restart.

wait_event	Description
ReplicationSlotRestoreSync	Flushes a streaming replication slot file to a disk during a restart.
ReplicationSlotSync	Flushes a temporary streaming replication slot file to a disk during checkpoint execution.
ReplicationSlotWrite	Writes a temporary streaming replication slot file during checkpoint execution.
SLRUFlushSync	Flushes the **pg_clog**, **pg_subtrans**, and **pg_multixact** files to a disk, mainly during checkpoint execution and database shutdown.
SLRURead	Reads the **pg_clog**, **pg_subtrans**, and **pg_multixact** files.
SLRUSync	Writes dirty pages into the **pg_clog**, **pg_subtrans**, and **pg_multixact** files, and flushes the files to a disk, mainly during checkpoint execution and database shutdown.
SLRUWrite	Writes the **pg_clog**, **pg_subtrans**, and **pg_multixact** files.
TimelineHistoryRead	Reads the timeline history file during database startup.
TimelineHistorySync	Flushes the timeline history file to a disk during database startup.
TimelineHistoryWrite	Writes to the timeline history file during database startup.
TwophaseFileRead	Reads the **pg_twophase** file, mainly during two-phase transaction commit and restoration.
TwophaseFileSync	Flushes the **pg_twophase** file to a disk, mainly during two-phase transaction commit and restoration.
TwophaseFileWrite	Writes the **pg_twophase** file, mainly during two-phase transaction commit and restoration.
WALBootstrapSync	Flushes an initialized WAL file to a disk during database initialization.
WALBootstrapWrite	Writes an initialized WAL file during database initialization.
WALCopyRead	Reads operation generated when an existing WAL file is read for replication after archiving and restoration.
WALCopySync	Flushes a replicated WAL file to a disk after archiving and restoration.

wait_event	Description
WALCopyWrite	Writes operation generated when an existing WAL file is read for replication after archiving and restoration.
WALInitSync	Flushes a newly initialized WAL file to a disk during log reclaiming or writing.
WALInitWrite	Initializes a newly created WAL file to **0** during log reclaiming or writing.
WALRead	Reads data from Xlogs during redo operations on two-phase files.
WALSyncMethodAssign	Flushes all open WAL files to a disk.
WALWrite	Writes a WAL file.
WALBufferAccess	Used to access WAL buffer. (To ensure performance, only the number of access times is counted in the kernel code, and the access duration is not counted.)
WALBufferFull	Used to write WAL files when the WAL buffer is full.
DoubleWriteFileRead	Doublewrites and reads a file.
DoubleWriteFileSync	Doublewrites a file and forcibly flushes files to disks.
DoubleWriteFileWrite	Doublewrites a file and writes a file.
PredoProcessPending	Waits for the replay of other records to complete during parallel log replay.
PredoApply	Waits for the replay of other threads to the LSN of the current thread during parallel log replay.
DisableConnectFileRead	Reads the HA lock fragment logic file.
DisableConnectFileSync	Forcibly flushes the HA lock fragment logic file to disks.
DisableConnectFileWrite	Writes the HA lock fragment logic file.
BufHashTableSearch	Searches the hash table in the shared buffer (page eviction may be triggered).
StrategyGetBuffer	Obtains pages from strategy buffer (page eviction may be triggered).
UndoFileExtend	Extends an undo file.
UndoFilePrefetch	Prefetches an undo file.
UndoFileRead	Reads an undo file.

wait_event	Description
UndoFileWrite	Writes an undo file.
UndoFileSync	Flushes an undo file to a disk.
UndoFileUnlink	Deletes an undo file.
UndoMetaSync	Flushes an undo metadata file to a disk.
DWSingleFlushGetPos	Searches for an available location for a single-page doublewrite file.
DWSingleFlushWrite	Flushes a single-page doublewrite file to a disk.
CkptWaitPageWriterFlush	Waits for page refreshing during full checkpoint execution.
CkptWaitPageWriterSync	Waits for the modified file to be synchronized to the disk before checkpointing.
CkptWaitCommitTransac-tionFinish	Waits until all transactions are committed before checkpointing.
MPFL_INIT	Initiates max_page_flush_lsn.
MPFL_READ	Reads max_page_flush_lsn.
MPFL_WRITE	Writes max_page_flush_lsn.
OBSList	Traverses an OBS directory.
OBSRead	Reads an OBS object.
OBSWrite	Writes an OBS object.
LOGCTRL_SLEEP	Waits for the standby node to catch up with logs.
ShareStorageWalRead	Reads log files from the shared disk.
ShareStorageWalWrite	Writes log files to the shared disk.
ShareStorageCtlInfoRead	Reads control information from the shared disk.
ShareStorageCtlInfoWrite	Writes control information to the shared disk.
SegFileExtend	Extends segment-page files.
SegReadDisk	Reads segment-page files.
SegWriteDisk	Writes segment-page files.
SegSync	Flushes segment-page files.
SegFileShrink	Collapses segment-page files.

The following table describes the corresponding wait events when **wait_status** is **acquire lock**.

Table 12-415 List of wait events corresponding to transaction locks

wait_event	Description
relation	Adds a lock to a table.
extend	Adds a lock to a table being scaled out.
partition	Adds a lock to a partitioned table.
partition_seq	Adds a lock to a partition of a partitioned table.
page	Adds a lock to a table page.
tuple	Adds a lock to a tuple on a page.
transactionid	Adds a lock to a transaction ID.
virtualxid	Adds a lock to a virtual transaction ID.
object	Adds a lock to an object.
userlock	Adds a lock to a user.
advisory	Adds an advisory lock.
filenode	Adds a lock to a file name and controls the concurrent sequence of file-level operations.
subtransactionid	Adds a lock to a sub-transaction ID.
tuple_uid	Adds a lock to the hidden column of a tuple UID.

12.3.12.188 PG_TIMEZONE_ABBREVS

PG_TIMEZONE_ABBREVS displays information about all available time zones.

Table 12-416 PG_TIMEZONE_ABBREVS columns

Name	Type	Description
abbrev	text	Time zone name abbreviation
utc_offset	interval	Offset from UTC
is_dst	boolean	Whether DST is used. If DST is used, the value is **TRUE**. Otherwise, the value is **FALSE**.

12.3.12.189 PG_TIMEZONE_NAMES

PG_TIMEZONE_NAMES displays all time zone names that can be recognized by **SET TIMEZONE**, along with their abbreviations, UTC offsets, and daylight saving time (DST) statuses.

Table 12-417 PG_TIMEZONE_NAMES columns

Name	Type	Description
name	text	Name of the time zone
abbrev	text	Abbreviation of the ime zone name
utc_offset	interval	Offset from UTC
is_dst	boolean	Whether DST is used. If DST is used, the value is **TRUE**. Otherwise, the value is **FALSE**.

12.3.12.190 PG_TOTAL_MEMORY_DETAIL

PG_TOTAL_MEMORY_DETAIL displays memory usage of a node in the database.

Table 12-418 PG_TOTAL_MEMORY_DETAIL columns

Name	Type	Description
nodename	text	Specifies the node name.
memorytype	text	Memory name
memorymbytes	integer	Size of the used memory in the unit of MB.

12.3.12.191 PG_TOTAL_USER_RESOURCE_INFO

PG_TOTAL_USER_RESOURCE_INFO displays resource usage of all users. Only administrators can query this view. This view is valid only when the GUC parameter **use_workload_manager** is set to **on**. I/O monitoring items are valid only when **enable_logical_io_statistics** is set to **on**.

Table 12-419 PG_TOTAL_USER_RESOURCE_INFO columns

Name	Type	Description
username	name	Username
used_memory	integer	Used memory, in MB
total_memory	integer	Available memory, in MB. The value **0** indicates that the available memory is not limited and depends on the maximum memory available in the database.
used_cpu	double precision	Number of CPU cores in use. CPU usage data is collected only in complex jobs, and the value is the CPU usage of the related Cgroup.

Name	Type	Description
total_cpu	integer	Total number of CPU cores of the Cgroup associated with the user on the node
used_space	bigint	Used permanent table storage space, in KB
total_space	bigint	Available permanent table storage space, in KB (**-1** if the storage space is not limited)
used_temp_space	bigint	Used temporary space, in KB
total_temp_space	bigint	Total available temporary space, in KB (**-1** if the temporary space is not limited)
used_spill_space	bigint	Size of the used operator flushing space, in KB
total_spill_space	bigint	Total size of the available operator flushing space, in KB (**-1** if the space is not limited)
read_kbytes	bigint	Primary database node: total bytes read by the user's complex jobs on all database nodes in the last 5 seconds, in KB Database node: total bytes read by the user's complex jobs from the instance startup time to the current time, in KB
write_kbytes	bigint	Primary database node: total bytes written by the user's complex jobs on all database nodes in the last 5 seconds, in KB Database node: total bytes written by the user's complex jobs from the instance startup time to the current time, in KB
read_counts	bigint	Primary database node: total number of read times of the user's complex jobs on all database nodes in the last 5 seconds Database node: total number of read times of the user's complex jobs from the instance startup time to the current time
write_counts	bigint	Primary database node: total number of write times of the user's complex jobs on all database nodes in the last 5 seconds Database node: total number of write times of the user's complex jobs from the instance startup time to the current time

Name	Type	Description
read_speed	double precision	Primary database node: average read rate of the user's complex jobs on a single database node in the last 5 seconds, in KB/s Database node: average read rate of the user's complex jobs on the database node in the last 5 seconds, in KB/s
write_speed	double precision	Primary database node: average write rate of the user's complex jobs on a single database node in the last 5 seconds, in KB/s Database node: average write rate of the user's complex jobs on the database node in the last 5 seconds, in KB/s

12.3.12.192 PG_TOTAL_USER_RESOURCE_INFO_OID

PG_TOTAL_USER_RESOURCE_INFO_OID displays resource usage of all users. Only administrators can query this view. This view is valid only when the GUC parameter **use_workload_manager** is set to **on**.

Table 12-420 PG_TOTAL_USER_RESOURCE_INFO_OID columns

Name	Type	Description
userid	oid	User ID.
used_memory	integer	Size of the memory being used, in MB.
total_memory	integer	Available memory (unit: MB) The value **0** indicates that the available memory is not limited and depends on the maximum memory available in the database.
used_cpu	double precision	Number of CPU cores in use.
total_cpu	integer	Total number of CPU cores of the Cgroup associated with the user on the node.
used_space	bigint	Used storage space, in KB.
total_space	bigint	Available storage space (unit: KB). The value **-1** indicates that the space is not limited.
used_temp_space	bigint	Used temporary storage space, in KB.
total_temp_space	bigint	Total available temporary space, in KB (**-1** if the temporary space is not limited).

Name	Type	Description
used_spill_space	bigint	Used disk space for spilling, in KB.
total_spill_space	bigint	Total available disk space for spilling, in KB. The value **-1** indicates that the space is not limited.
read_kbytes	bigint	Amount of data read from the disk, in KB.
write_kbytes	bigint	Amount of data written to the disk, in KB.
read_counts	bigint	Number of disk read times.
write_counts	bigint	Number of disk write times.
read_speed	double precision	Disk read rate, in B/ms.
write_speed	double precision	Disk write rate, in B/ms.

12.3.12.193 PG_VARIABLE_INFO

PG_VARIABLE_INFO records information about transaction IDs and OIDs of the current node in the database.

Table 12-421 PG_VARIABLE_INFO columns

Name	Type	Description
node_name	text	Node name
next_oid	oid	OID generated next time for the node
next_xid	xid	Transaction ID generated next time for the node
oldest_xid	xid	Oldest transaction ID on the node.
xid_vac_limit	xid	Critical point (transaction ID) that triggers forcible autovacuum
oldest_xid_db	oid	OID of the database that has the minimum datafrozenxid on the node
last_extend_csn_logpage	xid	Number of the last extended csnlog page
start_extend_csn_logpage	xid	Number of the page from which csnlog extending starts
next_commit_seqno	xid	CSN generated next time for the node
latest_completed_xid	xid	Latest transaction ID on the node after the transaction commission or rollback

Name	Type	Description
startup_max_xid	xid	Last transaction ID before the node is powered off

12.3.12.194 PG_VIEWS

PG_VIEWS provides access to basic information about each view in the database.

Table 12-422 PG_VIEWS columns

Name	Type	Reference	Description
schemaname	name	**PG_NAMESPACE**.nspname	Name of the schema that contains the view
viewname	name	**PG_CLASS**.relname	View name
viewowner	name	**PG_AUTHID**.Erolname	Owner of the view
definition	text	-	Definition of the view

12.3.12.195 PGXC_PREPARED_XACTS

PGXC_PREPARED_XACTS displays two-phase transactions in the **prepared** phase. Only users with the **system admin** or **monitor admin** permission can view the information.

Table 12-423 PGXC_PREPARED_XACTS columns

Name	Type	Description
pgxc_prepared_xact	text	Displays the two-phase transaction currently in the **prepared** phase.

12.3.12.196 PGXC_THREAD_WAIT_STATUS

This view is not supported in centralized scenarios.

12.3.12.197 PLAN_TABLE

PLAN_TABLE displays plan information collected by **EXPLAIN PLAN**. Plan information is in a session-level lifecycle. After a session exits, the data will be deleted. Data is isolated between sessions and between users. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 12-424 PLAN_TABLE columns

Name	Type	Description
statement_id	character varying(30)	Query tag specified by a user
plan_id	bigint	Query ID
id	integer	ID of each operator in a generated plan
operation	character varying(30)	Operation description of an operator in a plan
options	character varying(255)	Operation option
object_name	name	Object name corresponding to the operation, which is not the object alias used in the query. The object name is defined by users.
object_type	character varying(30)	Object type
object_owner	name	Schema to which the object belongs. It is defined by users.
projection	character varying(4000)	Returned column information
cost	double precision	Execution cost estimated by the optimizer for an operator
cardinality	double precision	Number of rows estimated by the optimizer for an operator
remarks	character varying(4000)	Not supported. Set it to **NULL**.
timestamp	timestamp(0) without time zone	Not supported. Set it to **NULL**.
object_node	character varying(128)	Not supported. Set it to **NULL**.
object_alias	character varying(261)	Not supported. Set it to **NULL**.
object_instance	numeric	Not supported. Set it to **NULL**.
optimizer	character varying(255)	Not supported. Set it to **NULL**.
search_columns	numeric	Not supported. Set it to **NULL**.
parent_id	numeric	Not supported. Set it to **NULL**.

Name	Type	Description
depth	numeric	Not supported. Set it to **NULL**.
position	numeric	Not supported. Set it to **NULL**.
bytes	numeric	Not supported. Set it to **NULL**.
other_tag	character varying(255)	Not supported. Set it to **NULL**.
partition_start	character varying(255)	Not supported. Set it to **NULL**.
partition_stop	character varying(255)	Not supported. Set it to **NULL**.
partition_id	numeric	Not supported. Set it to **NULL**.
other	character varying	Not supported. Set it to **NULL**.
other_xml	clob	Not supported. Set it to **NULL**.
distribution	character varying(20)	Not supported. Set it to **NULL**.
cpu_cost	numeric	Not supported. Set it to **NULL**.
io_cost	numeric	Not supported. Set it to **NULL**.
temp_space	numeric	Not supported. Set it to **NULL**.
access_predicates	character varying(4000)	Not supported. Set it to **NULL**.
filter_predicates	character varying(4000)	Not supported. Set it to **NULL**.
time	numeric	Not supported. Set it to **NULL**.
qblock_name	character varying(128)	Not supported. Set it to **NULL**.

NOTE

- A valid **object_type** value consists of a relkind type defined in **PG_CLASS** (**TABLE**, **INDEX**, **SEQUENCE**, **VIEW**, **COMPOSITE TYPE**, or **TOASTVALUE TOAST**) and the rtekind type used in the plan (**SUBQUERY**, **JOIN**, **FUNCTION**, **VALUES**, **CTE**, or **REMOTE_QUERY**).
- For RangeTableEntry (RTE), **object_owner** is the object description used in the plan. Non-user-defined objects do not have **object_owner**.
- Information in the **statement_id**, **object_name**, **object_owner**, and **projection** columns is stored in letter cases specified by users and information in other columns is stored in uppercase.
- **PLAN_TABLE** supports only **SELECT** and **DELETE** and does not support other DML operations.

12.3.12.198 SYS_DUMMY

SYS_DUMMY is automatically created by the database based on the data dictionary. It has only one text column in only one row for storing expression calculation results. This view is accessible to all users. This view exists in the PG_CATALOG and SYS schemas.

Table 12-425 SYS_DUMMY columns

Name	Type	Description
DUMMY	text	Expression calculation result

12.3.12.199 V_INSTANCE

V_INSTANCE displays instance information in current database. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 12-426 V_INSTANCE columns

Name	Type	Description
instance_number	oid	OID of the current database
instance_name	character varying(16)	Name of the current database
host_name	character varying(64)	Host name
version	character varying(17)	Not supported. Set it to **NULL**.
version_legacy	character varying(17)	Not supported. Set it to **NULL**.
version_full	character varying(17)	Not supported. Set it to **NULL**.
startup_time	timestamp(0) without time zone	Not supported. Set it to **NULL**.
status	character varying(12)	Not supported. Set it to **NULL**.
parallel	character varying(3)	Not supported. Set it to **NULL**.
thread#	numeric	Not supported. Set it to **NULL**.
archiver	character varying(7)	Not supported. Set it to **NULL**.

Name	Type	Description
log_switch_wait	character varying(15)	Not supported. Set it to **NULL**.
logins	character varying(10)	Not supported. Set it to **NULL**.
shutdown_pending	character varying(3)	Not supported. Set it to **NULL**.
database_status	character varying(17)	Not supported. Set it to **NULL**.
instance_role	character varying(18)	Not supported. Set it to **NULL**.
active_state	character varying(9)	Not supported. Set it to **NULL**.
blocked	character varying(3)	Not supported. Set it to **NULL**.
con_id	numeric	Not supported. Set it to **NULL**.
instance_mode	character varying(11)	Not supported. Set it to **NULL**.
edition	character varying(7)	Not supported. Set it to **NULL**.
family	character varying(80)	Not supported. Set it to **NULL**.
database_type	character varying(15)	Not supported. Set it to **NULL**.

12.3.12.200 V_MYSTAT

V_MYSTAT displays statistics information about all sessions in the database. Only system administrators can access this view. Common users can access this view only after being authorized. This view exists in both the **PG_CATALOG** and **SYS** schemas.

Table 12-427 V_MYSTAT columns

Name	Type	Description
sid	numeric	Current session ID
statistic#	numeric	Not supported. Set it to **NULL**.
value	numeric	Not supported. Set it to **NULL**.

Name	Type	Description
con_id	numeric	Not supported. Set it to **NULL**.

12.3.12.201 V_SESSION

V_SESSION describes information about all current sessions. Only system administrators can access this view. Common users can access this view only after being authorized. This view exists in both the PG_CATALOG and SYS schemas.

Table 12-428 V_SESSION columns

Name	Type	Description
saddr	raw	Not supported. Set it to **NULL**.
sid	bigint	Session ID
serial#	integer	Sequence number of the active background thread, which is **0** in GaussDB
audsid	numeric	Not supported. Set it to **NULL**.
paddr	raw	Not supported. Set it to **NULL**.
schema#	numeric	Not supported. Set it to **NULL**.
schemaname	name	Name of the user logged in to the backend
user#	oid	OID of the user that has logged in to the backend thread. The OID is **0** if the backend thread is a global auxiliary thread.
username	name	Name of the user logged in to the backend process. **username** is null if the backend thread is a global auxiliary thread.
command	numeric	Not supported. Set it to **NULL**.
ownerid	numeric	Not supported. Set it to **NULL**.
taddr	character varying(16)	Not supported. Set it to **NULL**.
lockwait	character varying(16)	Not supported. Set it to **NULL**.
machine	text	Host name of the connected client, as reported by a reverse DNS lookup of **client_addr**. This column will be non-null only for IP connections and only when **log_hostname** is enabled.
sql_id	bigint	Query ID

Name	Type	Description
client_info	text	Client information
event	text	Queuing status of a statement. Possible values are: • **waiting in queue**: The statement is in the queue. • **Empty**: The statement is running.
sql_exec_start	timestamp with time zone	Time when the currently active query was started, or time when the last query was started if the value of **state** is not **active**
program	text	Name of the application connected to the backend
status	text	Overall status of this backend Possible values are: • **active**: The backend is executing a query. • **idle**: The backend is waiting for a new client command. • **idle in transaction**: The backend is in a transaction, but there is no statement being executed in the transaction. • **idle in transaction** (aborted): The backend is in a transaction, but there are statements failed in the transaction. • **fastpath function call**: The backend is executing a fast-path function. • **disabled**: This state is reported if **track_activities** is disabled in this backend.
server	character varying(9)	Not supported. Set it to **NULL**.
pdml_status	character varying(8)	Specifies whether to enable a DML parallel execution in the current session.
port	numeric	Port number of the current session
process	character varying(24)	Process ID of the current session
logon_time	timestamp(0) without time zone	Login time of the current session
last_call_et	integer	Duration when the status of the current session changes last time
osuser	character varying(128)	Not supported. Set it to **NULL**.

Name	Type	Description
terminal	character varying (30)	Not supported. Set it to **NULL**.
type	character varying(10)	Not supported. Set it to **NULL**.
sql_address	raw	Not supported. Set it to **NULL**.
sql_hash_value	numeric	Not supported. Set it to **NULL**.
sql_child_number	numeric	Not supported. Set it to **NULL**.
sql_exec_id	numeric	Not supported. Set it to **NULL**.
prev_sql_addr	raw	Not supported. Set it to **NULL**.
prev_hash_value	numeric	Not supported. Set it to **NULL**.
prev_sql_id	character varying(13)	Not supported. Set it to **NULL**.
prev_child_number	numeric	Not supported. Set it to **NULL**.
prev_exec_start	timestamp(0) without time zone	Not supported. Set it to **NULL**.
prev_exec_id	numeric	Not supported. Set it to **NULL**.
plsql_entry_object_id	numeric	Not supported. Set it to **NULL**.
plsql_entry_subprogram_id	numeric	Not supported. Set it to **NULL**.
plsql_object_id	numeric	Not supported. Set it to **NULL**.
plsql_subprogram_id	numeric	Not supported. Set it to **NULL**.
module	character varying(64)	Not supported. Set it to **NULL**.
module_hash	numeric	Not supported. Set it to **NULL**.
action	character varying(64)	Not supported. Set it to **NULL**.

Name	Type	Description
action_hash	numeric	Not supported. Set it to **NULL**.
fixed_table_sequence	numeric	Not supported. Set it to **NULL**.
row_wait_obj#	numeric	Not supported. Set it to **NULL**.
row_wait_file#	numeric	Not supported. Set it to **NULL**.
row_wait_block#	numeric	Not supported. Set it to **NULL**.
row_wait_row#	numeric	Not supported. Set it to **NULL**.
top_level_call#	numeric	Not supported. Set it to **NULL**.
pdml_enabled	character varying(3)	Not supported. Set it to **NULL**.
failover_type	character varying(13)	Not supported. Set it to **NULL**.
failover_method	character varying(10)	Not supported. Set it to **NULL**.
failed_over	character varying(3)	Not supported. Set it to **NULL**.
resource_consumer_group	character varying(32)	Not supported. Set it to **NULL**.
pddl_status	character varying(8)	Not supported. Set it to **NULL**.
pq_status	character varying(8)	Not supported. Set it to **NULL**.
current_queue_duration	numeric	Not supported. Set it to **NULL**.
client_identifier	character varying(64)	Not supported. Set it to **NULL**.
blocking_session_status	character varying(11)	Not supported. Set it to **NULL**.
blocking_instance	numeric	Not supported. Set it to **NULL**.
blocking_session	numeric	Not supported. Set it to **NULL**.

Name	Type	Description
final_blocking_session_status	character varying(11)	Not supported. Set it to **NULL**.
final_blocking_instance	numeric	Not supported. Set it to **NULL**.
final_blocking_session	numeric	Not supported. Set it to **NULL**.
seq#	numeric	Not supported. Set it to **NULL**.
event#	numeric	Not supported. Set it to **NULL**.
p1text	character varying(64)	Not supported. Set it to **NULL**.
p1	numeric	Not supported. Set it to **NULL**.
p1raw	raw	Not supported. Set it to **NULL**.
p2text	character varying(64)	Not supported. Set it to **NULL**.
p2	numeric	Not supported. Set it to **NULL**.
p2raw	raw	Not supported. Set it to **NULL**.
p3text	character varying(64)	Not supported. Set it to **NULL**.
p3	numeric	Not supported. Set it to **NULL**.
p3raw	raw	Not supported. Set it to **NULL**.
wait_class_id	numeric	Not supported. Set it to **NULL**.
wait_class#	numeric	Not supported. Set it to **NULL**.
wait_class	character varying(64)	Not supported. Set it to **NULL**.
wait_time	numeric	Not supported. Set it to **NULL**.
seconds_in_wait	numeric	Not supported. Set it to **NULL**.
state	character varying(19)	Not supported. Set it to **NULL**.
wait_time_micro	numeric	Not supported. Set it to **NULL**.
time_remaining_micro	numeric	Not supported. Set it to **NULL**.

Name	Type	Description
time_since_last_wait_micro	numeric	Not supported. Set it to **NULL**.
service_name	character varying(64)	Not supported. Set it to **NULL**.
sql_trace	character varying(8)	Not supported. Set it to **NULL**.
sql_trace_waits	character varying(5)	Not supported. Set it to **NULL**.
sql_trace_binds	character varying(5)	Not supported. Set it to **NULL**.
sql_trace_plan_stats	character varying(10)	Not supported. Set it to **NULL**.
session_edition_id	numeric	Not supported. Set it to **NULL**.
creator_addr	raw	Not supported. Set it to **NULL**.
creator_serial#	numeric	Not supported. Set it to **NULL**.
ecid	character varying(64)	Not supported. Set it to **NULL**.
sql_translation_profile_id	numeric	Not supported. Set it to **NULL**.
pga_tunable_mem	numeric	Not supported. Set it to **NULL**.
shard_ddl_status	character varying(8)	Not supported. Set it to **NULL**.
con_id	numeric	Not supported. Set it to **NULL**.
external_name	character varying(1024)	Not supported. Set it to **NULL**.
plsql_debugger_connected	character varying(5)	Not supported. Set it to **NULL**.

12.3.12.202 V$GLOBAL_TRANSACTION

V$GLOBAL_TRANSACTION displays information about currently active global transactions. By default, only the system administrator can access this view.

Common users can access the view only after being authorized. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 12-429 V$GLOBAL_TRANSACTION columns

Name	Type	Description
formatid	numeric	Global transaction format identifier. Not supported. Set it to **NULL**.
globalid	raw	Global transaction identifier.
branchid	raw	Global transaction branch identifier. Not supported. Set it to **NULL**. Each individual transaction of a global transaction is called a branch.
branches	numeric	Total number of global transaction branches.
refcount	numeric	Number of siblings for the global transaction (must be the same as branches).
preparecount	numeric	Number of prepared global transaction branches. If the value of **system_view_version** is greater than **0** and no prepared global transaction branch exists, the value is **0**. Otherwise, the value is **NULL**.
state	character varying(38)	Status of the global transaction branch.
flags	numeric	Numeric representation of the status.
coupling	character varying(15)	Indicates whether the branch is free (`FREE`), loosely coupled (`LOOSELY COUPLED`), or tightly coupled (TIGHTLY COUPLED). Not supported. Set it to **NULL**.
con_id	numeric	ID of the container related to data. Not supported. Set it to **0**.

12.3.12.203 V$LOCK

V$LOCK displays information about locks held by open transactions. By default, only the system administrator can access the system view. Common users must be authorized to access the system view. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 12-430 V$LOCK columns

Name	Type	Description
addr	raw	Not supported. Set it to **NULL**.

Name	Type	Description
kaddr	raw	Not supported. Set it to **NULL**.
sid	number	ID of the session that owns the lock.
type	varchar(2)	TM or TX. • TM: corresponding relation lock in pg_locks. • TX: other locks.
id1	number	• TM lock: **relationId** corresponding to the object ID. • TX lock: not supported. Set to **0**.
id2	number	• TM lock: The default value is **0**. • TX lock: not supported. Set to **0**.
lmode	number	For details, see **Table 12-431**.
request	number	For details, see **Table 12-431**.
ctime	number	Not supported. Set it to **NULL**.
block	number	Specifies whether it is blocked by other sessions. The value **1** indicates yes, and the value **0** indicates no.
con_id	number	Not supported. Set it to **NULL**.
blocksid	bigint	ID of another thread blocked by the current thread.

Table 12-431 Lock mode

Level	GaussDB
-	NULL
0	INVALID
1	AccessShare
2	RowShare
3	RowExclusive
4	ShareUpdateExclusive
5	Share
6	ShareRowExclusive
7	Exclusive
8	AccessExclusive

12.3.12.204 V$NLS_PARAMETERS

V$NLS_PARAMETERS displays the National Language Support (NLS) parameters and parameter values configured for the database. This view exists in the **PG_CATALOG** and **SYS** schemas and all users can access this view.

Table 12-432 V$NLS_PARAMETERS columns

Name	Type	Description
parameter	character varying(64)	NLS parameter name.
value	character varying(64)	NLS parameter value.
con_id	numeric	Not supported. Set it to **0**.

12.3.12.205 V$OPEN_CURSOR

V$OPEN_CURSOR displays information about cursors opened by all current sessions. By default, only the system administrator can access the system view. Common users must be authorized to access the system view. This view exists in both PG_CATALOG and SYS schema.

Table 12-433 V$OPEN_CURSOR columns

Name	Type	Description
saddr	raw	Not supported. Its value is **NULL**.
sid	numeric	Session ID.
user_name	character varying(128)	Username.
address	raw	Not supported. Its value is **NULL**.
hash_value	numeric	Not supported. Its value is **NULL**.
sql_id	character varying(13)	Query statement ID.
sql_text	character varying(60)	First 60 bytes of the SQL text of a cursor.
last_sql_active_time	timestamp(0) without time zone	Not supported. Its value is **NULL**.

Name	Type	Description
sql_exec_id	numeric	Not supported. Its value is **NULL**.
cursor_type	character varying(64)	Cursor type. • **OPEN-PL/SQL**: opened PL/SQL cursor. • **OPEN**: other open cursors.
child_address	raw	Not supported. Its value is **NULL**.
con_id	numeric	Not supported. Its value is **NULL**.

12.3.12.206 V$SESSION_WAIT

V$SESSION_WAIT stores the current wait event or the last wait event of each session of each user. By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 12-434 V$SESSION_WAIT columns

Name	Type	Description
sid	numeric	Session ID, which is mapped to the **V$SESSION.SID** column.
seq#	numeric	Not supported. Set it to **NULL**.
event	character varying(64)	If the session is waiting, the resource or event that is waiting for is displayed. If the session is not waiting, the resource or event that is waiting for the last time is displayed.
p1text	character varying(64)	Not supported. Set it to **NULL**.
p1	numeric	Not supported. Set it to **NULL**.
p1raw	raw	Not supported. Set it to **NULL**.
p2text	character varying(64)	Not supported. Set it to **NULL**.
p2	numeric	Not supported. Set it to **NULL**.
p2raw	raw	Not supported. Set it to **NULL**.
p3text	character varying(64)	Not supported. Set it to **NULL**.
p3	numeric	Not supported. Set it to **NULL**.

Name	Type	Description
p3raw	raw	Not supported. Set it to **NULL**.
wait_class_id	numeric	Not supported. Set it to **NULL**.
wait_class#	numeric	Not supported. Set it to **NULL**.
wait_class	character varying(64)	Name of a wait event type
wait_time	numeric	If the session is waiting, the value is **0**. If the session is not waiting, the options are as follows: ● >0: duration of the last wait (unit: centisecond). ● **-1**: The duration of the last wait is less than a centisecond. ● **-2**: The **TIMED_STATISTICS** parameter is set to **false**. This column has been deprecated and replaced by **WAIT_TIME_MICRO** and **STATE**.
seconds_in_wait	numeric	If the session is waiting, the value is the amount of time to wait. If the session is not waiting, the value is the amount of time since the last wait started. This column has been deprecated and replaced by the **WAIT_TIME_MICRO** and **TIME_SINCE_LAST_WAIT_MICRO** columns.
state	character varying(19)	Waiting state. The options are as follows: ● **WAITING**: The session is waiting. ● **WAITED UNKNOWN TIME**: The duration of the last wait is unknown. This value is used when **TIMED_STATISTICS** is set to **false**. ● **WAITED SHORT TIME**: The duration of the last wait is less than a centisecond. ● **WAITED KNOWN TIME**: duration of the last wait specified in the **WAIT_TIME** column.
wait_time_micro	numeric	Wait time, in microseconds. If the session is waiting, the value is the time spent in the current wait. If the session is not waiting, the value is the time spent for the last wait.
time_remaining_micro	numeric	Not supported. Set it to **NULL**.

Name	Type	Description
time_since_last_wait_micro	numeric	Time elapsed since the last wait ended, in microseconds. If the session is waiting, the value is **0**.
con_id	numeric	Not supported. Set it to **0**.

12.3.12.207 V$SYSSTAT

V$SYSSTAT displays the resource usage of the entire database instance since the database instance starts to run. By default, only the initial user or monitoring administrator can access the database. Other users can access the database only after being granted the MONADMIN permission. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 12-435 V$SYSSTAT columns

Name	Type	Description
statistic#	numeric	Statistics ID.
name	character varying(64)	Statistical item name.
class	numeric	Not supported. Set it to **NULL**.
value	numeric	Statistical item value.
stat_id	numeric	Not supported. Set it to **NULL**.
con_id	numeric	Not supported. Set it to **0**.

12.3.12.208 V$SYSTEM_EVENT

V$SYSTEM_EVENT displays information about all the wait events (summary of each wait event since the instance is started). By default, only the system administrator can access this view. Common users can access the view only after being authorized. This view exists in the **PG_CATALOG** and **SYS** schemas.

Table 12-436 V$SYSTEM_EVENT columns

Name	Type	Description
event	character varying(64)	Wait event name
total_waits	numeric	Total number of wait events
total_timeouts	numeric	Total number of timeout events

Name	Type	Description
time_waited	numeric	Total time (in centiseconds) of wait events
average_wait	numeric	Average time (in centiseconds) of wait events
time_waited_micro	numeric	Total time (in microseconds) of wait events
total_waits_fg	numeric	Not supported. Set it to **NULL**.
total_timeouts_fg	numeric	Not supported. Set it to **NULL**.
time_waited_fg	numeric	Not supported. Set it to **NULL**.
average_wait_fg	numeric	Not supported. Set it to **NULL**.
time_waited_micro_fg	numeric	Not supported. Set it to **NULL**.
event_id	numeric	Not supported. Set it to **NULL**.
wait_class_id	numeric	Not supported. Set it to **NULL**.
wait_class#	numeric	Not supported. Set it to **NULL**.
wait_class	character varying(64)	Class name of a wait event
con_id	numeric	Not supported. Set it to **0**.

12.3.12.209 V$VERSION

V$VERSION displays the version number of the database. This view exists in the **PG_CATALOG** and **SYS** schemas and all users can access this view.

Table 12-437 V$VERSION columns

Name	Type	Description
banner	character varying(80)	Component name and version number
banner_full	character varying(160)	Database version and specific version number
banner_legacy	character varying(80)	Database version
con_id	numeric	Not supported. Set it to **0**.

12.4 Discarded

The following centralized system catalogs and views have been discarded in the latest version.

12.4.1 System Views

12.4.1.1 GET_GLOBAL_PREPARED_XACTS

This view is not supported in the centralized system.

12.4.1.2 GS_CLUSTER_RESOURCE_INFO

This view is not supported in the centralized system.

12.4.1.3 GS_WLM_WORKLOAD_RECORDS

This view is not supported in the centralized system.

13 Schemas

Table 13-1 describes the schemas of GaussDB.

NOTE

Do not create service data of users in a schema that provides functional interfaces, including but not limited to tables and functions (such as dbe_* and pkg_*).

Table 13-1 Schemas supported by GaussDB

Schema	Description
blockchain	Stores the user history table that is automatically created when a tamper-proof table is created in the ledger database.
db4ai	Manages data of different versions in AI training.
dbe_perf	Diagnoses performance issues and is also the data source of WDR snapshots. After a database is installed, only the initial user and monitoring administrator have permission to view views and functions in this schema by default.
dbe_pldebugger	Debugs PL/SQL functions and stored procedures.
snapshot	Manages data related to WDR snapshots. By default, the initial user or monitoring administrator can access the data.
sqladvisor	Is used for distribution key recommendation and is unavailable in centralized deployment.
sys	Provides the system information view APIs.
pg_catalog	Maintains system catalog information, including system catalogs and all built-in data types, functions, and operators.
pg_toast	Stores large objects (for internal use).

Schema	Description
public	Public schema, which is used to store public objects. If **search_path** is not specified and a schema with the same name exists, new tables (and other objects) are created in the schema by default. If no schema with the same name exists, new tables (and other objects) are automatically placed in this public schema.
pkg_service	Manages information about the package service.
pkg_util	Manages information about the package tool.
dbe_raw	Advanced function package dbe_raw, which is used to convert raw data, obtain substrings, and calculate the length.
dbe_session	Advanced function package dbe_session, which is used to set the value of a specified attribute and support user query and verification.
dbe_lob	Advanced function package dbe_lob, which is used to read, write, and copy large files (CLOB/BLOB).
dbe_match	Advanced function package dbe_match, which is used to compare character string similarity.
dbe_task	Advanced function package dbe_task, which is used to schedule job tasks, including submitting tasks, canceling tasks, synchronizing task status, and updating task information, so that the database can periodically execute specific tasks.
dbe_sql	Advanced function package dbe_sql, which is used to execute dynamic SQL statements and construct query and other commands during application running.
dbe_file	Advanced function package **dbe_file**, which is used to read, copy, write, delete, and rename external database files.
dbe_output	Advanced function package dbe_output, which is used to print output information.
dbe_random	Advanced function package dbe_random, which is used to generate random seeds and random numbers.
dbe_application_info	Advanced function package dbe_application_info, which is used for recording client information.
dbe_utility	Advanced function package **dbe_utility**, which is used to invoke the debugging tool in a stored procedure, for example, to print error stacks.

Schema	Description
dbe_scheduler	Advanced function package **dbe_scheduler**, which is used to create scheduled jobs and enable the database to periodically execute specified tasks through programs and schedules. You can also perform external database tasks by authorizing and providing certificates.
information_schema	Stores information about objects defined in the current database.
dbe_pldeveloper	Compiles and debugs user stored procedures.
dbe_xmlgen	Advanced function package **dbe_xmlgen**, which is used to convert query results into XML character strings.
dbe_stats	Advanced function package **dbe_stats**, which is used to manage statistics.
dbe_describe	Advanced function package **dbe_describe**, which is used to query parameter information about stored procedures or functions.
dbe_sql_util	SQL O&M function, including the O&M interface of SQL patches and plan management.

13.1 Information Schema

An information schema named INFORMATION_SCHEMA automatically exists in all databases. An information schema consists of a group of views that contain information about objects defined in the current database. The owner of this schema is the initial database user. However, all users have only the permission to use this schema and do not have the permission to create objects such as tables and functions.

Information schemas are compatible with PG, including constraint_table_usage, domain_constraints, domain_udt_usage, domains, enabled_roles, key_column_usage, parameters, referential_constraints, applicable_roles, administrable_role_authorizations, attributes, character_sets, check_constraint_routine_usage, check_constraints, collations, collation_character_set_applicability, column_domain_usage, column_privileges, column_udt_usage, columns, constraint_column_usage, role_column_grants, routine_privileges, role_routine_grants, routines, schemata, sequences, table_constraints, table_privileges, role_table_grants, tables, triggered_update_columns, triggers, udt_privileges, role_udt_grants, usage_privileges, role_usage_grants, user_defined_types, view_column_usage, view_routine_usage, view_table_usage, views, data_type_privileges, element_types, column_options, foreign_data_wrapper_options, foreign_data_wrappers, foreign_server_options, foreign_servers, foreign_table_options, foreign_tables, user_mapping_options, user_mappings, sql_features, sql_implementation_info, sql_languages, sql_packages, sql_parts, sql_sizing, and sql_sizing_profiles.

The following sections display only the views that are not listed in the preceding description.

13.1.1 _PG_FOREIGN_DATA_WRAPPERS

Displays information about a foreign-data wrapper, as shown in **Table 13-2**. Only the sysadmin user has the permission to view this view.

Table 13-2 _PG_FOREIGN_DATA_WRAPPERS columns

Name	Type	Description
oid	oid	OID of the foreign-data wrapper
fdwowner	oid	OID of the owner of the foreign-data wrapper
fdwoptions	text[]	Foreign-data wrapper specific option, expressed in a string in the format of *keyword*=*value*
foreign_data_wrapper_catalog	information_schema.sql_identifier	Name of the database where the foreign-data wrapper is located (always the current database)
foreign_data_wrapper_name	information_schema.sql_identifier	Name of the foreign-data wrapper
authorization_identifier	information_schema.sql_identifier	Role of the owner of the foreign-data wrapper
foreign_data_wrapper_language	information_schema.character_data	Programming language of the foreign-data wrapper

13.1.2 _PG_FOREIGN_SERVERS

Displays information about a foreign server, as described in **Table 13-3**. Only the sysadmin user has the permission to view this view.

Table 13-3 _PG_FOREIGN_SERVERS columns

Name	Type	Description
oid	oid	OID of the foreign server
srvoptions	text[]	Foreign server specific options, expressed in a string in the format of *keyword*=*value*

Name	Type	Description
foreign_server_catalog	information_schema.sql_identifier	Name of the database where the foreign server is located (always the current database)
foreign_server_name	information_schema.sql_identifier	Name of the foreign server
foreign_data_wrapper_catalog	information_schema.sql_identifier	Name of the database where the foreign-data wrapper is located (always the current database)
foreign_data_wrapper_name	information_schema.sql_identifier	Name of the foreign-data wrapper
foreign_server_type	information_schema.character_data	Type of the foreign server
foreign_server_version	information_schema.character_data	Version of the foreign server
authorization_identifier	information_schema.sql_identifier	Role of the owner of the foreign server

13.1.3 _PG_FOREIGN_TABLE_COLUMNS

Displays column information about a foreign table, as described in **Table 13-4**. Only the sysadmin user has the permission to view this view.

Table 13-4 _PG_FOREIGN_TABLE_COLUMNS columns

Name	Type	Description
nspname	name	Schema name
relname	name	Table name
attname	name	Column name
attfdwoptions	text[]	Attribute-level foreign data wrapper options, expressed in a string in the format of *keyword=value*

13.1.4 _PG_FOREIGN_TABLES

Stores information about all foreign tables defined in the current database, as described in **Table 13-5**, whereas displays information about foreign tables

accessible to the current user. Only the sysadmin user has the permission to view this view.

Table 13-5 _PG_FOREIGN_TABLES columns

Name	Type	Description
foreign_table_catalog	information_schema.sql_identifier	Name of the database where the foreign table is located (always the current database)
foreign_table_schema	name	Name of the schema that the foreign table is in
foreign_table_name	name	Name of the foreign table
ftoptions	text[]	Foreign table options
foreign_server_catalog	information_schema.sql_identifier	Name of the database where the foreign server is located (always the current database)
foreign_server_name	information_schema.sql_identifier	Name of the foreign server
authorization_identifier	information_schema.sql_identifier	Role of the owner

13.1.5 _PG_USER_MAPPINGS

Stores mappings from local users to remote users, as described in **Table 13-6**. Only the **sysadmin** user has the permission to view this view.

Table 13-6 _PG_USER_MAPPINGS columns

Name	Type	Description
oid	oid	OID of the mapping from the local user to a remote user
umoptions	text[]	User mapping specific options, expressed in a string in the format of *keyword*=*value*.
umuser	oid	OID of the local user being mapped (**0** if the user mapping is public).
authorization_identifier	information_schema.sql_identifier	Role of the local user.

Name	Type	Description
foreign_server_catalog	information_schema.sql_identifier	Name of the database where the foreign server is located (always the current database)
foreign_server_name	information_schema.sql_identifier	Name of the foreign server.
srvowner	information_schema.sql_identifier	Owner of the foreign server.

13.1.6 INFORMATION_SCHEMA_CATALOG_NAME

Displays the name of the current database, as described in **Table 13-7**.

Table 13-7 INFORMATION_SCHEMA_CATALOG_NAME columns

Name	Type	Description
catalog_name	information_schema.sql_identifier	Current database name.

13.2 DBE_PERF Schema

In the DBE_PERF schema, views are used to diagnose performance issues and are also the data source of WDR snapshots. After the database is installed, only the initial user and monitoring administrator have the permission for the DBE_PERF schema by default. To ensure forward compatibility, the permission on the DBE_PERF schema has no change before and after an upgrade. In the current version, all users are not allowed to create operators in this schema. Existing operators are not affected. Organization views are divided based on multiple dimensions, such as OS, instance, and memory. These views comply with the following naming rules:

- Views starting with **GLOBAL_**: Request data from database nodes and return the data without processing them.
- Views starting with **SUMMARY_**: Summarize data in the database. In most cases, data from database nodes (sometimes only the primary database node) is processed and collected.
- Views that do not start with **GLOBAL_** or **SUMMARY_**: Local views that do not request data from other database nodes.

13.2.1 OS

13.2.1.1 OS_RUNTIME

Displays the running status of the current OS, as shown in **Table 13-8**.

Table 13-8 OS_RUNTIME columns

Name	Type	Description
id	integer	ID.
name	text	Name of the OS running status.
value	numeric	Value of the OS running status.
comments	text	Remarks of the OS running status.
cumulative	boolean	Specifies whether the value of the OS running status is cumulative.

13.2.1.2 GLOBAL_OS_RUNTIME

Provides OS running status information about all normal nodes in the database, as shown in **Table 13-9**.

Table 13-9 GLOBAL_OS_RUNTIME columns

Name	Type	Description
node_name	name	Node name
id	integer	ID
name	text	Name of the OS running status
value	numeric	Value of the OS running status
comments	text	Remarks of the OS running status
cumulative	boolean	Whether the value of the OS running status is cumulative

13.2.1.3 OS_THREADS

Provides status information about all threads on the current node, as shown in **Table 13-10**.

Table 13-10 OS_THREADS columns

Name	Type	Description
node_name	text	Name of the current node.
pid	bigint	ID of the thread running under the current node process.

Name	Type	Description
lwpid	integer	Lightweight thread ID corresponding to **pid**.
thread_name	text	Name of the thread corresponding to **pid**.
creation_time	timestamp with time zone	Creation time of the thread corresponding to **pid**.

13.2.1.4 GLOBAL_OS_THREADS

Provides status information about threads on all normal nodes in the database, as shown in **Table 13-11**.

Table 13-11 GLOBAL_OS_THREADS columns

Name	Type	Description
node_name	text	Node name
pid	bigint	ID of the thread running under the current node process
lwpid	integer	Lightweight thread ID corresponding to **pid**
thread_name	text	Name of the thread corresponding to **pid**
creation_time	timestamp with time zone	Creation time of the thread corresponding to **pid**

13.2.1.5 NODE_NAME

Provides the names of all normal nodes in the database, as shown in **Table 13-12**.

Table 13-12 NODE_NAME columns

Name	Type	Description
node_name	name	Node name

13.2.1.6 PERF_QUERY

Provides the name, tree structure, and percentage of the stack information collected on the current node, as shown in **Table 13-13**. The sysadmin or monadmin permission is required.

Table 13-13 PERF_QUERY columns

Name	Type	Description
backtrace	text	Text of the stack information tree structure.
overhead	double precision	Percentage of the time occupied by the current stack information in the entire stack collection process.

13.2.2 Instance

13.2.2.1 INSTANCE_TIME

Provides various time consumption information under the current database node, as shown in **Table 13-14**.

Table 13-14 INSTANCE_TIME columns

Name	Type	Description
stat_id	integer	Statistics ID
stat_name	text	Type name • DB_TIME: effective time spent by jobs in multi-core scenarios • CPU_TIME: CPU time cost. • EXECUTION_TIME: time spent in the executor. • PARSE_TIME: time spent on parsing SQL statements • PLAN_TIME: time spent on generating plans • REWRITE_TIME: time spent on rewriting SQL statements • PL_EXECUTION_TIME: execution time of the PL/SQL stored procedure • PL_COMPILATION_TIME: compilation time of the PL/SQL stored procedure • NET_SEND_TIME: time spent on the network • DATA_IO_TIME: time spent on I/Os.
value	bigint	Time value (unit: μs)

13.2.2.2 GLOBAL_INSTANCE_TIME

Provides various time consumption information under all normal nodes in the database, as shown in **Table 13-15**.

Table 13-15 GLOBAL_INSTANCE_TIME columns

Name	Type	Description
node_name	name	Node name
stat_id	integer	Statistics ID
stat_name	text	Type name. See the INSTANCE_TIME view.
value	bigint	Duration (unit: μs)

13.2.3 Memory

13.2.3.1 GLOBAL_MEMORY_NODE_DETAIL

Displays the memory usage of all normal nodes in the database, as shown in **Table 13-16**.

Table 13-16 GLOBAL_MEMORY_NODE_DETAIL columns

Name	Type	Description
nodename	text	Node name

Name	Type	Description
memorytype	text	Memory name. • **max_process_memory**: maximum available memory of the database node. • **process_used_memory**: memory occupied by a process • **max_dynamic_memory**: maximum dynamic memory • **dynamic_used_memory**: used dynamic memory • **dynamic_peak_memory**: dynamic peak value of the memory • **dynamic_used_shrctx**: context of the used dynamic shared memory. • **dynamic_peak_shrctx**: dynamic peak value of the shared memory context • **max_shared_memory**: maximum shared memory • **shared_used_memory**: used shared memory • **max_sctpcomm_memory**: maximum memory allowed for TCP proxy communication • **sctpcomm_used_memory**: used memory for TCP proxy communication • **sctpcomm_peak_memory**: peak memory of TCP proxy communication • **other_used_memory**: other used memory • **gpu_max_dynamic_memory**: maximum dynamic GPU memory • **gpu_dynamic_used_memory**: used dynamic GPU memory • **gpu_dynamic_peak_memory**: dynamic peak GPU memory • **pooler_conn_memory**: applied memory in the connection pool • **pooler_freeconn_memory**: memory occupied by idle connections in the connection pool • **storage_compress_memory**: memory used by the storage module for compression • **udf_reserved_memory**: reserved memory for the UDF
memorymbytes	integer	Size of the used memory in the unit of MB.

13.2.3.2 GLOBAL_SHARED_MEMORY_DETAIL

Displays the usage of shared memory contexts on all normal nodes in the database, as shown in **Table 13-17**.

Table 13-17 GLOBAL_SHARED_MEMORY_DETAIL columns

Name	Type	Description
node_name	name	Node name
contextname	text	Name of the memory context
level	smallint	Level of the memory context
parent	text	Name of the parent memory context
totalsize	bigint	Total size of the shared memory (unit: byte)
freesize	bigint	Remaining size of the shared memory (unit: byte)
usedsize	bigint	Used size of the shared memory (unit: byte)

13.2.3.3 MEMORY_NODE_DETAIL

Displays memory usage of a node in the database, as shown in **Table 13-18**.

Table 13-18 MEMORY_NODE_DETAIL columns

Name	Type	Description
nodename	text	Node name

Name	Type	Description
memorytype	text	Memory name • **max_process_memory**: maximum available memory of the database node. • **process_used_memory**: memory occupied by a process • **max_dynamic_memory**: maximum dynamic memory • **dynamic_used_memory**: used dynamic memory • **dynamic_peak_memory**: dynamic peak value of the memory • **dynamic_used_shrctx**: context of the used dynamic shared memory. • **dynamic_peak_shrctx**: dynamic peak value of the shared memory context • **max_shared_memory**: maximum shared memory • **shared_used_memory**: used shared memory • **max_sctpcomm_memory**: maximum memory allowed for TCP proxy communication • **sctpcomm_used_memory**: used memory for TCP proxy communication • **sctpcomm_peak_memory**: memory peak of TCP proxy communication • **other_used_memory**: other used memory • **gpu_max_dynamic_memory**: maximum dynamic GPU memory • **gpu_dynamic_used_memory**: used dynamic memory of GPU • **gpu_dynamic_peak_memory**: dynamic peak value of the GPU memory • **pooler_conn_memory**: applied memory in the connection pool • **pooler_freeconn_memory**: memory occupied by idle connections in the connection pool • **storage_compress_memory**: memory used by the storage module for compression • **udf_reserved_memory**: reserved memory for the UDF
memorymbytes	integer	Size of the used memory in the unit of MB.

13.2.3.4 SHARED_MEMORY_DETAIL

Displays the usage information about shared memory contexts on the current node, as shown in **Table 13-19**.

Table 13-19 Table 1 SHARED_MEMORY_DETAIL columns

Name	Type	Description
contextname	text	Name of the memory context
level	smallint	Level of the memory context
parent	text	Name of the parent memory context
totalsize	bigint	Total size of the shared memory (unit: byte)
freesize	bigint	Remaining size of the shared memory (unit: byte)
usedsize	bigint	Used size of the shared memory (unit: byte)

13.2.4 File

13.2.4.1 FILE_IOSTAT

Records statistics about data file I/Os to indicate I/O performance and detect performance problems such as abnormal I/O operations, as shown in **Table 13-20**.

Table 13-20 FILE_IOSTAT columns

Name	Type	Description
filenum	oid	File identifier
dbid	oid	Database ID
spcid	oid	Tablespace ID
phyrds	bigint	Number of times of reading physical files
phywrts	bigint	Number of times of writing into physical files
phyblkrd	bigint	Number of times of reading physical file blocks
phyblkwrt	bigint	Number of times of writing into physical file blocks

Name	Type	Description
readtim	bigint	Total duration of reading (unit: μs)
writetim	bigint	Total duration of writing (unit: μs)
avgiotim	bigint	Average duration of reading and writing (unit: μs)
lstiotim	bigint	Duration of the last file reading (unit: μs)
miniotim	bigint	Minimum duration of reading and writing (unit: μs)
maxiowtm	bigint	Maximum duration of reading and writing (unit: μs)

13.2.4.2 SUMMARY_FILE_IOSTAT

Records statistics about data file I/Os in the database to indicate I/O performance and detect performance problems such as abnormal I/O operations, as shown in **Table 13-21**.

The values of **phyrds**, **phywrts**, **phyblkrd**, **phyblkwrt**, **readtim** and **writetim** are summed up based on the data of each node. **avgiotim** is the average value (total duration/total times) of each node. **lstiotim** and **maxiowtm** are the maximum values of each node. **miniotim** is the minimum value of each node.

Table 13-21 SUMMARY_FILE_IOSTAT columns

Name	Type	Description
filenum	oid	File ID
dbid	oid	Database ID
spcid	oid	Tablespace ID
phyrds	numeric	Number of times of reading physical files
phywrts	numeric	Number of times of writing into physical files
phyblkrd	numeric	Number of times of reading physical file blocks
phyblkwrt	numeric	Number of times of writing into physical file blocks
readtim	numeric	Total duration of reading (unit: μs)
writetim	numeric	Total duration of writing (unit: μs)

Name	Type	Description
avgiotim	bigint	Average duration of reading and writing (unit: μs)
lstiotim	bigint	Duration of the last file reading (unit: μs)
miniotim	bigint	Minimum duration of reading and writing (unit: μs)
maxiowtm	bigint	Maximum duration of reading and writing (unit: μs)

13.2.4.3 GLOBAL_FILE_IOSTAT

Displays the data file I/O statistics of all nodes in the database, as described in Table 13-22.

Table 13-22 GLOBAL_FILE_IOSTAT columns

Name	Type	Description
node_name	name	Node name.
filenum	oid	File ID.
dbid	oid	Database ID.
spcid	oid	Tablespace ID.
phyrds	bigint	Number of times of reading physical files.
phywrts	bigint	Number of times of writing into physical files.
phyblkrd	bigint	Number of times of reading physical file blocks.
phyblkwrt	bigint	Number of times of writing into physical file blocks.
readtim	bigint	Total duration of reading (unit: μs).
writetim	bigint	Total duration of writing (unit: μs).
avgiotim	bigint	Average duration of reading and writing (unit: μs).
lstiotim	bigint	Duration of the last file reading (unit: μs).
miniotim	bigint	Minimum duration of reading and writing (unit: μs).

Name	Type	Description
maxiowtm	bigint	Maximum duration of reading and writing (unit: μs).

13.2.4.4 FILE_REDO_IOSTAT

Records statistics about redo logs (WALs) on the current node, as shown in **Table 13-23**.

Table 13-23 FILE_REDO_IOSTAT columns

Name	Type	Description
phywrts	bigint	Number of times writing into the WAL buffer
phyblkwrt	bigint	Number of blocks written into the WAL buffer
writetim	bigint	Duration of writing into Xlog files (unit: μs)
avgiotim	bigint	Average duration of writing into Xlog files (unit: μs). **avgiotim** = **writetim**/ **phywrts**
lstiotim	bigint	Duration of the last writing into Xlog files (unit: μs)
miniotim	bigint	Minimum duration of writing into Xlog files (unit: μs)
maxiowtm	bigint	Maximum duration of writing into Xlog files (unit: μs)

13.2.4.5 SUMMARY_FILE_REDO_IOSTAT

Records redo log (WAL) statistics of all nodes in the database, as shown in **Table 13-24**. The values of **phywrts**, **phyblkwrt**, and **writetim** are summed up based on the data of each node. **avgiotim** is the average value (summed up **writetim** or **phywrts**) of each node. **lstiotim** and **maxiowtm** are the maximum values of each node. **miniotim** is the minimum value of each node.

Table 13-24 SUMMARY_FILE_REDO_IOSTAT columns

Name	Type	Description
phywrts	numeric	Number of times writing into the WAL buffer

Name	Type	Description
phyblkwrt	numeric	Number of blocks written into the WAL buffer
writetim	numeric	Duration of writing into Xlog files (unit: μs)
avgiotim	bigint	Average duration of writing into Xlog files (unit: μs). **avgiotim** = **writetim**/**phywrts**
lstiotim	bigint	Duration of the last writing into Xlog files (unit: μs)
miniotim	bigint	Minimum duration of writing into Xlog files (unit: μs)
maxiowtm	bigint	Maximum duration of writing into Xlog files (unit: μs)

13.2.4.6 GLOBAL_FILE_REDO_IOSTAT

Displays statistics about redo logs (WALs) on nodes in the database, as shown in **Table 13-25**.

Table 13-25 GLOBAL_FILE_REDO_IOSTAT columns

Name	Type	Description
node_name	name	Node name
phywrts	bigint	Number of times writing into the WAL buffer
phyblkwrt	bigint	Number of blocks written into the WAL buffer
writetim	bigint	Duration of writing into Xlog files (unit: μs)
avgiotim	bigint	Average duration of writing into Xlog files (unit: μs). **avgiotim** = **writetim**/**phywrts**
lstiotim	bigint	Duration of the last writing into Xlog files (unit: μs)
miniotim	bigint	Minimum duration of writing into Xlog files (unit: μs)
maxiowtm	bigint	Maximum duration of writing into Xlog files (unit: μs)

13.2.4.7 LOCAL_REL_IOSTAT

Displays the accumulated I/O status of all data files on the current node, as shown in **Table 13-26**.

Table 13-26 LOCAL_REL_IOSTAT columns

Name	Type	Description
phyrds	bigint	Number of times of reading physical files
phywrts	bigint	Number of times of writing into physical files
phyblkrd	bigint	Number of times of reading physical file blocks
phyblkwrt	bigint	Number of times of writing into physical file blocks

13.2.4.8 GLOBAL_REL_IOSTAT

Displays statistics about data file I/Os on all nodes, as shown in **Table 13-27**.

Table 13-27 GLOBAL_REL_IOSTAT columns

Name	Type	Description
node_name	name	Node name
phyrds	bigint	Number of times of reading physical files
phywrts	bigint	Number of times of writing into physical files
phyblkrd	bigint	Number of times of reading physical file blocks
phyblkwrt	bigint	Number of times of writing into physical file blocks

13.2.4.9 SUMMARY_REL_IOSTAT

Obtains statistics about data file I/Os on all nodes, as shown in **Table 13-28**.

Table 13-28 SUMMARY_REL_IOSTAT columns

Name	Type	Description
phyrds	numeric	Number of times of reading physical files
phywrts	numeric	Number of times of writing into physical files
phyblkrd	numeric	Number of times of reading physical file blocks
phyblkwrt	numeric	Number of times of writing into physical file blocks

13.2.5 Object

13.2.5.1 STAT_USER_TABLES

Displays the status information about user-defined ordinary tables in all schemas on the current node, as described in **Table 13-29**.

Table 13-29 STAT_USER_TABLES columns

Name	Type	Description
relid	oid	OID of the table.
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on the table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.

Name	Type	Description
n_tup_hot_upd	bigint	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).
n_live_tup	bigint	Estimated number of live rows.
n_dead_tup	bigint	Estimated number of inactive rows.
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting **VACUUM FULL**).
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Last time at which the table was manually analyzed.
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.
vacuum_count	bigint	Number of times that the table has been manually vacuumed (not counting **VACUUM FULL**).
autovacuum_count	bigint	Number of times this table has been vacuumed by the autovacuum daemon.
analyze_count	bigint	Number of times the table has been manually analyzed.
autoanalyze_count	bigint	Number of times the table has been analyzed by the autovacuum daemon thread.
last_updated	timestamp with time zone	Time when the monitoring data of the object in the view is updated for the last time.

13.2.5.2 SUMMARY_STAT_USER_TABLES

Displays the sum of the status information about user-defined ordinary tables in all schemas on each database node, as described in **Table 13-30**. The timestamp column is not summed up and only the latest value of this column on all nodes is used.

Table 13-30 SUMMARY_STAT_USER_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	numeric	Number of sequential scans initiated on this table.
seq_tup_read	numeric	Number of live rows fetched by sequential scans.
idx_scan	numeric	Number of index scans initiated on the table.
idx_tup_fetch	numeric	Number of live rows fetched by index scans.
n_tup_ins	numeric	Number of rows inserted.
n_tup_upd	numeric	Number of rows updated.
n_tup_del	numeric	Number of rows deleted.
n_tup_hot_upd	numeric	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).
n_live_tup	numeric	Estimated number of live rows.
n_dead_tup	numeric	Estimated number of inactive rows.
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting **VACUUM FULL**).
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Last time when the table was manually analyzed.
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.
vacuum_count	numeric	Number of times that the table has been manually vacuumed (not counting **VACUUM FULL**).
autovacuum_count	numeric	Number of times that the table has been vacuumed by the autovacuum daemon.

Name	Type	Description
analyze_count	numeric	Number of times that the table has been manually analyzed.
autoanalyze_count	numeric	Number of times that the table has been analyzed by the autovacuum daemon thread.

13.2.5.3 GLOBAL_STAT_USER_TABLES

Displays the status information (not summed up) about user-defined common tables in all schemas on each database node, as described in **Table 13-31**.

Table 13-31 GLOBAL_STAT_USER_TABLES columns

Name	Type	Description
node_name	name	Node name.
relid	oid	OID of the table.
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on this table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).
n_live_tup	bigint	Estimated number of live rows.
n_dead_tup	bigint	Estimated number of inactive rows.

Name	Type	Description
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting **VACUUM FULL**).
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Last time at which this table was manually analyzed.
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.
vacuum_count	bigint	Number of times that the table has been manually vacuumed (not counting **VACUUM FULL**).
autovacuum_count	bigint	Number of times that the table has been vacuumed by the autovacuum daemon.
analyze_count	bigint	Number of times that the table has been manually analyzed.
autoanalyze_count	bigint	Number of times that the table has been analyzed by the autovacuum daemon thread.

13.2.5.4 STAT_USER_INDEXES

Displays the index status information about the user-defined ordinary tables on the current node, as shown in **Table 13-32**.

Table 13-32 STAT_USER_INDEXES columns

Name	Type	Description
relid	oid	OID of the table for this index.
indexrelid	oid	OID of the index.
schemaname	name	Name of the schema where the index is located.
relname	name	Name of the table for the index.
indexrelname	name	Index name.
idx_scan	bigint	Number of index scans performed on the index.

Name	Type	Description
idx_tup_read	bigint	Number of index entries returned by scans on the index.
idx_tup_fetch	bigint	Number of live rows fetched in the original table by a simple index scan that uses the index.
last_updated	timestamp with time zone	Time when the monitoring data of the object in the view is updated for the last time.

13.2.5.5 SUMMARY_STAT_USER_INDEXES

Displays the sum of the index status information about user-defined ordinary tables in all schemas on each database node, as described in **Table 13-33**.

Table 13-33 SUMMARY_STAT_USER_INDEXES columns

Name	Type	Description
schemaname	name	Name of the schema where the index is located.
relname	name	Name of the table for the index.
indexrelname	name	Index name.
idx_scan	numeric	Number of index scans performed on the index.
idx_tup_read	numeric	Number of index entries returned by scans on the index.
idx_tup_fetch	numeric	Number of active rows fetched in the original table by a simple index scan that uses the index.

13.2.5.6 GLOBAL_STAT_USER_INDEXES

Displays the index status information about user-defined common tables in all schemas on each database node, as described in **Table 13-34**. Data on different nodes is not summed up.

Table 13-34 GLOBAL_STAT_USER_INDEXES columns

Name	Type	Description
node_name	name	Node name.
relid	oid	OID of the table for this index.
indexrelid	oid	OID of the index.

Name	Type	Description
schemaname	name	Name of the schema where the index is located.
relname	name	Name of the table for the index.
indexrelname	name	Index name.
idx_scan	bigint	Number of index scans performed on the index.
idx_tup_read	bigint	Number of index entries returned by scans on the index.
idx_tup_fetch	bigint	Number of active rows fetched in the original table by a simple index scan that uses the index.

13.2.5.7 STAT_SYS_TABLES

Displays status information about all the system catalogs in the pg_catalog, information_schema, and pg_toast schemas on the current node, as described in **Table 13-35**.

Table 13-35 STAT_SYS_TABLES columns

Name	Type	Description
relid	oid	OID of the table.
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on the table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.

Name	Type	Description
n_tup_hot_upd	bigint	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).
n_live_tup	bigint	Estimated number of live rows.
n_dead_tup	bigint	Estimated number of inactive rows.
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting **VACUUM FULL**).
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Last time at which the table was manually analyzed.
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.
vacuum_count	bigint	Number of times that the table has been manually vacuumed (not counting **VACUUM FULL**).
autovacuum_count	bigint	Number of times this table has been vacuumed by the autovacuum daemon.
analyze_count	bigint	Number of times the table has been manually analyzed.
autoanalyze_count	bigint	Number of times the table has been analyzed by the autovacuum daemon thread.
last_updated	timestamp with time zone	Time when the monitoring data of the object in the view is updated for the last time.

13.2.5.8 SUMMARY_STAT_SYS_TABLES

Displays the sum of the status information in all system catalogs on each node in pg_catalog, information_schema, and pg_toast schemas. The status information in the system catalogs on each node is summed up, as described in **Table 13-36**. Columns of the timestamp type are not summed up, and only the latest values of these columns on all nodes are used.

Table 13-36 SUMMARY_STAT_SYS_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	numeric	Number of sequential scans initiated on this table.
seq_tup_read	numeric	Number of live rows fetched by sequential scans.
idx_scan	numeric	Number of index scans initiated on the table.
idx_tup_fetch	numeric	Number of live rows fetched by index scans.
n_tup_ins	numeric	Number of rows inserted.
n_tup_upd	numeric	Number of rows updated.
n_tup_del	numeric	Number of rows deleted.
n_tup_hot_upd	numeric	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).
n_live_tup	numeric	Estimated number of live rows.
n_dead_tup	numeric	Estimated number of inactive rows.
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting **VACUUM FULL**).
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Last time when the table was manually analyzed.
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.
vacuum_count	numeric	Number of times that the table has been manually vacuumed (not counting **VACUUM FULL**).
autovacuum_count	numeric	Number of times that the table has been vacuumed by the autovacuum daemon.

Name	Type	Description
analyze_count	numeric	Number of times that the table has been manually analyzed.
autoanalyze_count	numeric	Number of times that the table has been analyzed by the autovacuum daemon thread.

13.2.5.9 GLOBAL_STAT_SYS_TABLES

Displays the status information about all system catalogs in pg_catalog, information_schema, and pg_toast schemas on each database node, as described in **Table 13-37**. Data on different nodes is not summed up.

Table 13-37 GLOBAL_STAT_SYS_TABLES columns

Name	Type	Description
node_name	name	Node name.
relid	oid	OID of the table.
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on this table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).
n_live_tup	bigint	Estimated number of live rows.
n_dead_tup	bigint	Estimated number of inactive rows.

Name	Type	Description
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting **VACUUM FULL**).
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Last time when the table was manually analyzed.
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.
vacuum_count	bigint	Number of times that the table has been manually vacuumed (not counting **VACUUM FULL**).
autovacuum_count	bigint	Number of times that the table has been vacuumed by the autovacuum daemon.
analyze_count	bigint	Number of times that the table has been manually analyzed.
autoanalyze_count	bigint	Number of times that the table has been analyzed by the autovacuum daemon thread.

13.2.5.10 STAT_SYS_INDEXES

Displays index status information about all the system catalogs in the pg_catalog, information_schema, and pg_toast schemas, as shown in **Table 13-38**.

Table 13-38 STAT_SYS_INDEXES columns

Name	Type	Description
relid	oid	OID of the table for this index.
indexrelid	oid	OID of the index.
schemaname	name	Name of the schema where the index is located.
relname	name	Name of the table that the index is created for.
indexrelname	name	Index name.
idx_scan	bigint	Number of index scans performed on the index.

Name	Type	Description
idx_tup_read	bigint	Number of index entries returned by scans on the index.
idx_tup_fetch	bigint	Number of live rows fetched in the original table by a simple index scan that uses the index.
last_updated	timestamp with time zone	Time when the monitoring data of the object in the view is updated for the last time.

13.2.5.11 SUMMARY_STAT_SYS_INDEXES

Displays the summary of the index status information of all system catalogs in the pg_catalog, information_schema, and pg_toast schemas on each database node, as shown in **Table 13-39**.

Table 13-39 SUMMARY_STAT_SYS_INDEXES columns

Name	Type	Description
schemaname	name	Name of the schema where the index is located.
relname	name	Name of the table for the index
indexrelname	name	Index name
idx_scan	numeric	Number of index scans performed on the index.
idx_tup_read	numeric	Number of index entries returned by scans on the index.
idx_tup_fetch	numeric	Number of live rows fetched in the original table by a simple index scan that uses the index.

13.2.5.12 GLOBAL_STAT_SYS_INDEXES

Displays the index status information about all system catalogs in the pg_catalog, information_schema, and pg_toast schemas on each database node, as shown in **Table 13-40**. Data on different nodes is not summed up.

Table 13-40 GLOBAL_STAT_SYS_INDEXES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the table for this index
indexrelid	oid	OID of the index.

Name	Type	Description
schemaname	name	Name of the schema where the index is located.
relname	name	Name of the table for the index.
indexrelname	name	Index name
idx_scan	bigint	Number of index scans performed on the index.
idx_tup_read	bigint	Number of index entries returned by scans on the index.
idx_tup_fetch	bigint	Number of live rows fetched in the original table by a simple index scan that uses the index.

13.2.5.13 STAT_ALL_TABLES

Displays the status of each table (including the TOAST table) on the current node of the database, as described in **Table 13-41**.

Table 13-41 STAT_ALL_TABLES columns

Name	Type	Description
relid	oid	OID of the table.
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on the table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).

Name	Type	Description
n_live_tup	bigint	Estimated number of live rows.
n_dead_tup	bigint	Estimated number of inactive rows.
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting **VACUUM FULL**).
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Last time at which the table was manually analyzed.
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.
vacuum_count	bigint	Number of times that the table has been manually vacuumed (not counting **VACUUM FULL**).
autovacuum_count	bigint	Number of times this table has been vacuumed by the autovacuum daemon.
analyze_count	bigint	Number of times the table has been manually analyzed.
autoanalyze_count	bigint	Number of times the table has been analyzed by the autovacuum daemon thread.
last_updated	timestamp with time zone	Time when the monitoring data of the object in the view is updated for the last time.

13.2.5.14 SUMMARY_STAT_ALL_TABLES

Displays the sum of the status information of each table (including the TOAST table) on each node of the database, as described in **Table 13-42**. The timestamp column is not summed up and only the latest value of this column on all nodes is used.

Table 13-42 SUMMARY_STAT_ALL_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.

Name	Type	Description
seq_scan	numeric	Number of sequential scans initiated on this table.
seq_tup_read	numeric	Number of live rows fetched by sequential scans.
idx_scan	numeric	Number of index scans initiated on the table.
idx_tup_fetch	numeric	Number of live rows fetched by index scans.
n_tup_ins	numeric	Number of rows inserted.
n_tup_upd	numeric	Number of rows updated.
n_tup_del	numeric	Number of rows deleted.
n_tup_hot_upd	numeric	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).
n_live_tup	numeric	Estimated number of live rows.
n_dead_tup	numeric	Estimated number of inactive rows.
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting **VACUUM FULL**).
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Last time when the table was manually analyzed.
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.
vacuum_count	numeric	Number of times that the table has been manually vacuumed (not counting **VACUUM FULL**).
autovacuum_count	numeric	Number of times that the table has been vacuumed by the autovacuum daemon.
analyze_count	numeric	Number of times that the table has been manually analyzed.
autoanalyze_count	numeric	Number of times that the table has been analyzed by the autovacuum daemon thread.

Name	Type	Description
last_updated	timestamp with time zone	Time when the monitoring data of the table in the view is updated for the last time.

13.2.5.15 GLOBAL_STAT_ALL_TABLES

Displays the status of each table (including the TOAST table) on each node of the database, as described in **Table 13-43**. Data on different nodes is not summed up.

Table 13-43 GLOBAL_STAT_ALL_TABLES columns

Name	Type	Description
node_name	name	Node name.
relid	oid	OID of the table.
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on this table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of rows updated by HOT (that is, the number of rows whose index columns are not updated).
n_live_tup	bigint	Estimated number of live rows.
n_dead_tup	bigint	Estimated number of inactive rows.
last_vacuum	timestamp with time zone	Last time at which this table was manually vacuumed (not counting **VACUUM FULL**).

Name	Type	Description
last_autovacuum	timestamp with time zone	Last time when the table was vacuumed by the autovacuum daemon thread.
last_analyze	timestamp with time zone	Last time when the table was manually analyzed.
last_autoanalyze	timestamp with time zone	Last time when the table was analyzed by the autovacuum daemon thread.
vacuum_count	bigint	Number of times that the table has been manually vacuumed (not counting **VACUUM FULL**).
autovacuum_count	bigint	Number of times that the table has been vacuumed by the autovacuum daemon.
analyze_count	bigint	Number of times that the table has been manually analyzed.
autoanalyze_count	bigint	Number of times that the table has been analyzed by the autovacuum daemon thread.
last_updated	timestamp with time zone	Time when the monitoring data of the table in the view is updated for the last time.

13.2.5.16 STAT_ALL_INDEXES

Displays the access information about each index on the current node of the database, as shown in **Table 13-44**.

Table 13-44 STAT_ALL_INDEXES columns

Name	Type	Description
relid	oid	OID of the table for this index
indexrelid	oid	OID of the index.
schemaname	name	Name of the schema where the index is located.
relname	name	Name of the table for the index.
indexrelname	name	Index name
idx_scan	bigint	Number of index scans performed on the index.
idx_tup_read	bigint	Number of index entries returned by scans on the index.

Name	Type	Description
idx_tup_fetch	bigint	Number of live rows fetched in the original table by a simple index scan that uses the index.
last_updated	timestamp with time zone	Time when the monitoring data of the object in the view is updated for the last time.

13.2.5.17 SUMMARY_STAT_ALL_INDEXES

Displays the sum of the access information of each index on each database node, as described in **Table 13-45**. The timestamp column is not summed up and only the latest value of this column on all nodes is used.

Table 13-45 SUMMARY_STAT_ALL_INDEXES columns

Name	Type	Description
schemaname	name	Name of the schema where the index is located.
relname	name	Name of the table for the index.
indexrelname	name	Index name.
idx_scan	numeric	Number of index scans performed on the index.
idx_tup_read	numeric	Number of index entries returned by scans on the index.
idx_tup_fetch	numeric	Number of active rows fetched in the original table by a simple index scan that uses the index.
last_updated	timestamp with time zone	Time when the monitoring data of the index in the view is updated for the last time.

13.2.5.18 GLOBAL_STAT_ALL_INDEXES

Displays the access information about each index on each node of the database, as described in **Table 13-46**. The status information of each index on each node is not summed up.

Table 13-46 GLOBAL_STAT_ALL_INDEXES columns

Name	Type	Description
node_name	name	Node name.
relid	oid	OID of the table for this index.

Name	Type	Description
indexrelid	oid	OID of the index.
schemaname	name	Name of the schema where the index is located.
relname	name	Name of the table for the index.
indexrelname	name	Index name.
idx_scan	bigint	Number of index scans performed on the index.
idx_tup_read	bigint	Number of index entries returned by scans on the index.
idx_tup_fetch	bigint	Number of active rows fetched in the original table by a simple index scan that uses the index.
last_updated	timestamp with time zone	Time when the monitoring data of the index in the view is updated for the last time.

13.2.5.19 STAT_DATABASE

Displays the statistics of the current node in the database, as shown in **Table 13-47**.

Table 13-47 STAT_DATABASE columns

Name	Type	Description
datid	oid	OID of a database
datname	name	Database name
numbackends	integer	Number of backends currently connected to this database.
xact_commit	bigint	Number of transactions in this database that have been committed
xact_rollback	bigint	Number of transactions in this database that have been rolled back
blks_read	bigint	Number of disk blocks read in this database
blks_hit	bigint	Number of disk blocks that have been hit in the cache. In this case, data does not need to be read from disks. (The cache includes only the database buffer cache and does not include the file system cache of the OS.)

Name	Type	Description
tup_returned	bigint	Number of live rows fetched by sequential scans and number of index rows returned by index scans in the database.
tup_fetched	bigint	Number of rows returned by the current database through indexes.
tup_inserted	bigint	Number of rows inserted.
tup_updated	bigint	Number of rows updated.
tup_deleted	bigint	Number of rows deleted.
conflicts	bigint	Number of queries canceled due to conflicts with database replay (conflicts occur only on the standby node). For details, see **STAT_DATABASE_CONFLICTS**.
temp_files	bigint	Number of temporary files created by queries in this database. All temporary files are counted, which is not affected by the value of the GUC parameter **log_temp_files**.
temp_bytes	bigint	Total amount of data written to temporary files by queries in this database. All temporary files are counted, which is not affected by the value of the GUC parameter **log_temp_files**.
deadlocks	bigint	Number of deadlocks detected in this database
blk_read_time	double precision	Time spent reading data file blocks by backends in this database (unit: ms)
blk_write_time	double precision	Time spent reading data file blocks by backends in this database (unit: ms)
stats_reset	timestamp with time zone	Time at which the current statistics were reset

13.2.5.20 SUMMARY_STAT_DATABASE

Displays the summary of the status statistics of each database node, as described in **Table 13-48**. The timestamp column is not summed up and only the latest value of this column on all nodes is used.

Table 13-48 SUMMARY_STAT_DATABASE

Name	Type	Description
datname	name	Database name.
numbackends	bigint	Number of backends currently connected to this database.
xact_commit	numeric	Number of transactions in this database that have been committed.
xact_rollback	numeric	Number of transactions in this database that have been rolled back.
blks_read	numeric	Number of disk blocks read in this database.
blks_hit	numeric	Number of disk blocks that have been hit in the cache. In this case, data does not need to be read from disks. (The cache includes only the database buffer cache and does not include the file system cache of the OS.)
tup_returned	numeric	Number of live rows fetched by sequential scans and number of index rows returned by index scans in the database.
tup_fetched	numeric	Number of rows returned by the current database through indexes.
tup_inserted	bigint	Number of rows inserted.
tup_updated	bigint	Number of rows updated.
tup_deleted	bigint	Number of rows deleted.
conflicts	bigint	Number of queries canceled due to conflicts with database replay (conflicts occur only on the standby node). For details, see **STAT_DATABASE_CONFLICTS**.
temp_files	numeric	Number of temporary files created by queries in this database. All temporary files are counted, which is not affected by the value of the GUC parameter **log_temp_files**.
temp_bytes	numeric	Total amount of data written to temporary files by queries in this database. All temporary files are counted, which is not affected by the value of the GUC parameter **log_temp_files**.

Name	Type	Description
deadlocks	bigint	Number of deadlocks detected in this database.
blk_read_time	double precision	Time spent reading data file blocks by backends in this database (unit: ms).
blk_write_time	double precision	Time spent writing into data file blocks by backends in this database (unit: ms).
stats_reset	timestamp with time zone	Time at which the current statistics were reset.

13.2.5.21 GLOBAL_STAT_DATABASE

Displays the statistics of each database node, as shown in **Table 13-49**. The database status information on different nodes is not summed up.

Table 13-49 GLOBAL_STAT_DATABASE columns

Name	Type	Description
node_name	name	Node name.
datid	oid	OID of the database
datname	name	Name of the database
numbackends	integer	Number of backends currently connected to this database.
xact_commit	bigint	Number of transactions in this database that have been committed.
xact_rollback	bigint	Number of transactions in this database that have been rolled back.
blks_read	bigint	Number of disk blocks read in this database.
blks_hit	bigint	Number of disk blocks that have been hit in the cache. In this case, data does not need to be read from disks. (The cache includes only the database buffer cache and does not include the file system cache of the OS.)
tup_returned	bigint	Number of live rows fetched by sequential scans and number of index rows returned by index scans in the database.
tup_fetched	bigint	Number of rows returned by the current database through indexes.

Name	Type	Description
tup_inserted	bigint	Number of rows inserted.
tup_updated	bigint	Number of rows updated.
tup_deleted	bigint	Number of rows deleted.
conflicts	bigint	Number of queries canceled due to conflicts with database replay (conflicts occur only on the standby node). For details, see **STAT_DATABASE_CONFLICTS**.
temp_files	bigint	Number of temporary files created by queries in this database. All temporary files are counted, which is not affected by the value of the GUC parameter **log_temp_files**.
temp_bytes	bigint	Total amount of data written to temporary files by queries in this database. All temporary files are counted, which is not affected by the value of the GUC parameter **log_temp_files**.
deadlocks	bigint	Number of deadlocks detected in this database
blk_read_time	double precision	Time spent reading data file blocks by backends in this database (unit: ms)
blk_write_time	double precision	Time spent writing into data file blocks by backends in this database (unit: ms)
stats_reset	timestamp with time zone	Time at which the current statistics were reset

13.2.5.22 STAT_DATABASE_CONFLICTS

Displays statistics about the conflict status of the current database node, as shown in **Table 13-50**.

Table 13-50 STAT_DATABASE_CONFLICTS columns

Name	Type	Description
datid	oid	Database ID
datname	name	Database name
confl_tablespace	bigint	Number of conflicting tablespaces

Name	Type	Description
confl_lock	bigint	Number of conflicting locks
confl_snapshot	bigint	Number of conflicting snapshots
confl_bufferpin	bigint	Number of conflicting buffers
confl_deadlock	bigint	Number of conflicting deadlocks

13.2.5.23 SUMMARY_STAT_DATABASE_CONFLICTS

Displays the summay of the conflict status statistics of each node in the database, as shown in **Table 13-51**.

Table 13-51 SUMMARY_STAT_DATABASE_CONFLICTS columns

Name	Type	Description
datname	name	Database name
confl_tablespace	bigint	Number of conflicting tablespaces
confl_lock	bigint	Number of conflicting locks
confl_snapshot	bigint	Number of conflicting snapshots
confl_bufferpin	bigint	Number of conflicting buffers
confl_deadlock	bigint	Number of conflicting deadlocks

13.2.5.24 GLOBAL_STAT_DATABASE_CONFLICTS

Displays the conflict status statistics of each database node, as shown in **Table 13-52**. The status information of each database node is not summed up.

Table 13-52 GLOBAL_STAT_DATABASE_CONFLICTS columns

Name	Type	Description
node_name	name	Node name.
datid	oid	Database ID
datname	name	Database name
confl_tablespace	bigint	Number of conflicting tablespaces
confl_lock	bigint	Number of conflicting locks

Name	Type	Description
confl_snapshot	bigint	Number of conflicting snapshots
confl_bufferpin	bigint	Number of conflicting buffers
confl_deadlock	bigint	Number of conflicting deadlocks

13.2.5.25 STAT_XACT_ALL_TABLES

Displays transaction status information about all ordinary tables and TOAST tables in all schemas on the current node, as shown in **Table 13-53**.

Table 13-53 STAT_XACT_ALL_TABLES columns

Name	Type	Description
relid	oid	Table OID.
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on the table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of HOT updated rows (that is, the number of rows whose index columns are not updated).

13.2.5.26 SUMMARY_STAT_XACT_ALL_TABLES

Displays the summary of transaction status information about all ordinary tables and TOAST tables in all schemas of each database node, as shown in **Table 13-54**.

Table 13-54 SUMMARY_STAT_XACT_ALL_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	numeric	Number of sequential scans initiated on this table
seq_tup_read	numeric	Number of live rows fetched by sequential scans
idx_scan	numeric	Number of index scans initiated on the table
idx_tup_fetch	numeric	Number of live rows fetched by index scans
n_tup_ins	numeric	Number of rows inserted
n_tup_upd	numeric	Number of rows updated
n_tup_del	numeric	Number of rows deleted
n_tup_hot_upd	numeric	Number of HOT updated rows (that is, the number of rows whose index columns are not updated).

13.2.5.27 GLOBAL_STAT_XACT_ALL_TABLES

Displays the transaction status information about all ordinary tables and TOAST tables in all schemas of each database node, as shown in **Table 13-55**. The transaction status information about tables on different nodes is not summed up.

Table 13-55 GLOBAL_STAT_XACT_ALL_TABLES columns

Name	Type	Description
node_name	name	Node name.
relid	oid	OID of the table
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on this table
seq_tup_read	bigint	Number of live rows fetched by sequential scans
idx_scan	bigint	Number of index scans initiated on the table
idx_tup_fetch	bigint	Number of live rows fetched by index scans
n_tup_ins	bigint	Number of rows inserted

Name	Type	Description
n_tup_upd	bigint	Number of rows updated
n_tup_del	bigint	Number of rows deleted
n_tup_hot_upd	bigint	Number of HOT updated rows (that is, the number of rows whose index columns are not updated).

13.2.5.28 STAT_XACT_SYS_TABLES

Displays transaction status information about the system catalogs in the schemas of the current node, as shown in **Table 13-56**.

Table 13-56 STAT_XACT_SYS_TABLES columns

Name	Type	Description
relid	oid	Table OID.
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on the table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of HOT updated rows (that is, the number of rows whose index columns are not updated).

13.2.5.29 SUMMARY_STAT_XACT_SYS_TABLES

Displays the summary of the transaction status information about system catalogs in all the schemas of each database node, as shown in **Table 13-57**.

Table 13-57 SUMMARY_STAT_XACT_SYS_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	numeric	Number of sequential scans initiated on this table
seq_tup_read	numeric	Number of live rows fetched by sequential scans.
idx_scan	numeric	Number of index scans initiated on the table.
idx_tup_fetch	numeric	Number of live rows fetched by index scans.
n_tup_ins	numeric	Number of rows inserted.
n_tup_upd	numeric	Number of rows updated.
n_tup_del	numeric	Number of rows deleted.
n_tup_hot_upd	numeric	Number of HOT updated rows (that is, the number of rows whose index columns are not updated).

13.2.5.30 GLOBAL_STAT_XACT_SYS_TABLES

Displays the transaction status information about the system catalogs in all the schemas of each database node, as shown in **Table 13-58**. The transaction status information of the tables on different nodes is not summed up.

Table 13-58 GLOBAL_STAT_XACT_SYS_TABLES columns

Name	Type	Description
node_name	name	Node name.
relid	oid	Table OID.
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on this table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.

Name	Type	Description
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of HOT updated rows (that is, the number of rows whose index columns are not updated).

13.2.5.31 STAT_XACT_USER_TABLES

Displays transaction status information about the user tables in schemas on the current node, as shown in **Table 13-59**.

Table 13-59 STAT_XACT_USER_TABLES columns

Name	Type	Description
relid	oid	Table OID.
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on the table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of HOT updated rows (that is, the number of rows whose index columns are not updated).

13.2.5.32 SUMMARY_STAT_XACT_USER_TABLES

Displays the summary of the transaction status information about user tables in all the schemas of each database node, as shown in **Table 13-60**.

Table 13-60 SUMMARY_STAT_XACT_USER_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	numeric	Number of sequential scans initiated on this table.
seq_tup_read	numeric	Number of live rows fetched by sequential scans.
idx_scan	numeric	Number of index scans initiated on the table.
idx_tup_fetch	numeric	Number of live rows fetched by index scans.
n_tup_ins	numeric	Number of rows inserted.
n_tup_upd	numeric	Number of rows updated.
n_tup_del	numeric	Number of rows deleted.
n_tup_hot_upd	numeric	Number of HOT updated rows (that is, the number of rows whose index columns are not updated).

13.2.5.33 GLOBAL_STAT_XACT_USER_TABLES

Displays the transaction status information about the user tables in all the schemas of each database node, as shown in **Table 13-61**. The transaction status information of the tables on different nodes is not summed up.

Table 13-61 GLOBAL_STAT_XACT_USER_TABLES columns

Name	Type	Description
node_name	name	Node name.
relid	oid	Table OID.
schemaname	name	Name of the schema where the table is located.
relname	name	Table name.
seq_scan	bigint	Number of sequential scans initiated on this table.
seq_tup_read	bigint	Number of live rows fetched by sequential scans.
idx_scan	bigint	Number of index scans initiated on the table.
idx_tup_fetch	bigint	Number of live rows fetched by index scans.
n_tup_ins	bigint	Number of rows inserted.

Name	Type	Description
n_tup_upd	bigint	Number of rows updated.
n_tup_del	bigint	Number of rows deleted.
n_tup_hot_upd	bigint	Number of HOT updated rows (that is, the number of rows whose index columns are not updated).

13.2.5.34 STAT_XACT_USER_FUNCTIONS

Displays statistics about function executions in the current transaction on the current database node, as shown in **Table 13-62**.

Table 13-62 STAT_XACT_USER_FUNCTIONS columns

Name	Type	Description
funcid	oid	OID of a function.
schemaname	name	Name of the schema where the function is located.
funcname	name	Function name.
calls	bigint	Number of times the function has been called.
total_time	double precision	Total time spent in this function and all other functions called by it.
self_time	double precision	Time spent in this function, excluding other functions called by it.

13.2.5.35 SUMMARY_STAT_XACT_USER_FUNCTIONS

Displays the summary of function execution statistics in a transaction on each database node, as shown in **Table 13-63**.

Table 13-63 SUMMARY_STAT_XACT_USER_FUNCTIONS columns

Name	Type	Description
schemaname	name	Name of the schema where the function is located.
funcname	name	Function name
calls	numeric	Number of times that the function has been called

Name	Type	Description
total_time	double precision	Total time spent in this function and all other functions called by it.
self_time	double precision	Time spent in this function, excluding other functions called by it.

13.2.5.36 GLOBAL_STAT_XACT_USER_FUNCTIONS

Displays statistics about function execution in a transaction on each database node, as shown in **Table 13-64**. Data on different nodes is not summed up.

Table 13-64 GLOBAL_STAT_XACT_USER_FUNCTIONS columns

Name	Type	Description
node_name	name	Node name.
funcid	oid	OID of a function.
schemaname	name	Name of the schema where the function is located.
funcname	name	Function name.
calls	bigint	Number of times that the function has been called.
total_time	double precision	Total time spent in this function and all other functions called by it.
self_time	double precision	Time spent in this function, excluding other functions called by it.

13.2.5.37 STAT_BAD_BLOCK

Obtains the information about table and index read failures on the current node, as shown in **Table 13-65**.

Table 13-65 STAT_BAD_BLOCK columns

Name	Type	Description
nodename	text	Node name.
databaseid	integer	Database OID.
tablespaceid	integer	Tablespace OID.
relfilenode	integer	Relationship file node.

Name	Type	Description
bucketid	smallint	ID of the bucket for consistent hashing.
forknum	integer	Fork number.
error_count	integer	Number of errors.
first_time	timestamp with time zone	Time when the page damage occurs for the first time.
last_time	timestamp with time zone	Time when the page damage occurs for the last time.

13.2.5.38 SUMMARY_STAT_BAD_BLOCK

Obtains the summary of the table and index read failure information of each database node in a period specified by **first_time** and **last_time**, as shown in **Table 13-66**. The value of **first_time** is the start time and the value of **last_time** is the end time.

Table 13-66 SUMMARY_STAT_BAD_BLOCK columns

Name	Type	Description
databaseid	integer	Database OID.
tablespaceid	integer	Tablespace OID.
relfilenode	integer	Relationship file node.
forknum	bigint	Fork number.
error_count	bigint	Number of errors.
first_time	timestamp with time zone	Time when the page damage occurs for the first time.
last_time	timestamp with time zone	Time when the page damage occurs for the last time.

13.2.5.39 GLOBAL_STAT_BAD_BLOCK

Obtains the table and index read failure information of each database node, as shown in **Table 13-67**. The read failure information of different nodes is not summed up.

Table 13-67 GLOBAL_STAT_BAD_BLOCK columns

Name	Type	Description
node_name	text	Node name.

Name	Type	Description
databaseid	integer	Database OID.
tablespaceid	integer	Tablespace OID.
relfilenode	integer	Relationship file node.
forknum	integer	Fork number.
error_count	integer	Number of errors.
first_time	timestamp with time zone	Time when the page damage occurs for the first time.
last_time	timestamp with time zone	Time when the page damage occurs for the last time.

13.2.5.40 STAT_USER_FUNCTIONS

Displays statistics about user-defined functions (the function language is not an internal language) in all the schemas of the current node, as shown in **Table 13-68**.

Table 13-68 STAT_USER_FUNCTIONS columns

Name	Type	Description
funcid	oid	OID of a function.
schemaname	name	Schema name.
funcname	name	Rename the customized function.
calls	bigint	Number of times that the function has been called.
total_time	double precision	Total time spent in this function, including other functions called by it (unit: ms).
self_time	double precision	Time spent in this function, excluding other functions called by it (unit: ms).

13.2.5.41 SUMMARY_STAT_USER_FUNCTIONS

Displays the summary of statistics about user-defined functions on each database node, as shown in **Table 13-69**.

Table 13-69 SUMMARY_STAT_USER_FUNCTIONS columns

Name	Type	Description
schemaname	name	Schema name.
funcname	name	UDF name.
calls	numeric	Number of times the function has been called.
total_time	double precision	Total time spent in this function, including other functions called by it (unit: ms).
self_time	double precision	Time spent in this function, excluding other functions called by it (unit: ms).

13.2.5.42 GLOBAL_STAT_USER_FUNCTIONS

Displays the statistics of UDFs on each database node, as shown in **Table 13-70**. The statistics on different nodes are not summed up.

Table 13-70 GLOBAL_STAT_USER_FUNCTIONS columns

Name	Type	Description
node_name	name	Node name.
funcid	oid	ID of the function.
schemaname	name	Name of the schema where the function is located.
funcname	name	UDF name.
calls	bigint	Number of times the function has been called.
total_time	double precision	Total time spent in this function, including other functions called by it (unit: ms).
self_time	double precision	Time spent in this function, excluding other functions called by it (unit: ms).

13.2.6 Workload

13.2.6.1 WORKLOAD_SQL_COUNT

Displays the distribution of SQL statements in workloads on the current node, as shown in **Table 13-71**.

Table 13-71 WORKLOAD_SQL_COUNT columns

Name	Type	Description
workload	name	Workload name
select_count	bigint	Number of **SELECT** statements
update_count	bigint	Number of **UPDATE** statements
insert_count	bigint	Number of **INSERT** statements
delete_count	bigint	Number of **DELETE** statements
ddl_count	bigint	Number of **DDL** statements
dml_count	bigint	Number of **DML** statements
dcl_count	bigint	Number of **DCL** statements

13.2.6.2 SUMMARY_WORKLOAD_SQL_COUNT

Displays the distribution of SQL statements in workloads on each primary database node in the database, as shown in **Table 13-72**.

Table 13-72 SUMMARY_WORKLOAD_SQL_COUNT columns

Name	Type	Description
node_name	name	Node name.
workload	name	Workload name.
select_count	bigint	Number of SELECT statements.
update_count	bigint	Number of UPDATE statements.
insert_count	bigint	Number of INSERT statements.
delete_count	bigint	Number of DELETE statements.
ddl_count	bigint	Number of DDL statements.
dml_count	bigint	Number of DML statements.
dcl_count	bigint	Number of DCL statements.

13.2.6.3 WORKLOAD_TRANSACTION

Displays information about transactions loaded on the current node, as shown in **Table 13-73**.

Table 13-73 WORKLOAD_TRANSACTION columns

Name	Type	Description
workload	name	Workload name.
commit_counter	bigint	Number of user transactions committed.
rollback_counter	bigint	Number of user transactions rolled back.
resp_min	bigint	Minimum response time of user transactions (unit: μs).
resp_max	bigint	Maximum response time of user transactions (unit: μs).
resp_avg	bigint	Average response time of user transactions (unit: μs).
resp_total	bigint	Total response time of user transactions (unit: μs).
bg_commit_counter	bigint	Number of background transactions committed.
bg_rollback_counter	bigint	Number of background transactions rolled back.
bg_resp_min	bigint	Minimum response time of background transactions (unit: μs).
bg_resp_max	bigint	Maximum response time of background transactions (unit: μs).
bg_resp_avg	bigint	Average response time of background transactions (unit: μs).
bg_resp_total	bigint	Total response time of background transactions (unit: μs).

13.2.6.4 SUMMARY_WORKLOAD_TRANSACTION

Displays information about transactions loaded in the database, as shown in **Table 13-74**.

Table 13-74 SUMMARY_WORKLOAD_TRANSACTION columns

Name	Type	Description
workload	name	Workload name.

Name	Type	Description
commit_counter	numeric	Number of user transactions committed.
rollback_counter	numeric	Number of user transactions rolled back.
resp_min	bigint	Minimum response time of user transactions (unit: μs).
resp_max	bigint	Maximum response time of user transactions (unit: μs).
resp_avg	bigint	Average response time of user transactions (unit: μs).
resp_total	numeric	Total response time of user transactions (unit: μs).
bg_commit_counter	numeric	Number of background transactions committed.
bg_rollback_counter	numeric	Number of background transactions rolled back.
bg_resp_min	bigint	Minimum response time of background transactions (unit: μs).
bg_resp_max	bigint	Maximum response time of background transactions (unit: μs).
bg_resp_avg	bigint	Average response time of background transactions (unit: μs).
bg_resp_total	numeric	Total response time of background transactions (unit: μs).

13.2.6.5 GLOBAL_WORKLOAD_TRANSACTION

Displays load information about workloads on each node, as shown in **Table 13-75**.

Table 13-75 GLOBAL_WORKLOAD_TRANSACTION columns

Name	Type	Description
node_name	name	Node name.
workload	name	Workload name.
commit_counter	bigint	Number of user transactions committed.
rollback_counter	bigint	Number of user transactions rolled back.
resp_min	bigint	Minimum response time of user transactions (unit: μs).

Name	Type	Description
resp_max	bigint	Maximum response time of user transactions (unit: μs).
resp_avg	bigint	Average response time of user transactions (unit: μs).
resp_total	bigint	Total response time of user transactions (unit: μs).
bg_commit_counter	bigint	Number of background transactions committed.
bg_rollback_counter	bigint	Number of background transactions rolled back.
bg_resp_min	bigint	Minimum response time of background transactions (unit: μs).
bg_resp_max	bigint	Maximum response time of background transactions (unit: μs).
bg_resp_avg	bigint	Average response time of background transactions (unit: μs).
bg_resp_total	bigint	Total response time of background transactions (unit: μs).

13.2.6.6 WORKLOAD_SQL_ELAPSE_TIME

WORKLOAD_SQL_ELAPSE_TIME collects SELECT, UPDATE, INSERT, and DELETE (SUID) statistics, as described in **Table 13-76**.

Table 13-76 WORKLOAD_SQL_ELAPSE_TIME columns

Name	Type	Description
workload	name	Workload name.
total_select_elapse	bigint	Total response time of SELECT statements (unit: μs).
max_select_elapse	bigint	Maximum response time of SELECT statements (unit: μs).
min_select_elapse	bigint	Minimum response time of SELECT statements (unit: μs).
avg_select_elapse	bigint	Average response time of SELECT statements (unit: μs).
total_update_elapse	bigint	Total response time of UPDATE statements (unit: μs).

Name	Type	Description
max_update_elapse	bigint	Maximum response time of UPDATE statements (unit: μs).
min_update_elapse	bigint	Minimum response time of UPDATE statements (unit: μs).
avg_update_elapse	bigint	Average response time of UPDATE statements (unit: μs).
total_insert_elapse	bigint	Total response time of INSERT statements (unit: μs).
max_insert_elapse	bigint	Maximum response time of INSERT statements (unit: μs).
min_insert_elapse	bigint	Minimum response time of INSERT statements (unit: μs).
avg_insert_elapse	bigint	Average response time of INSERT statements (unit: μs).
total_delete_elapse	bigint	Total response time of DELETE statements (unit: μs).
max_delete_elapse	bigint	Maximum response time of DELETE statements (unit: μs).
min_delete_elapse	bigint	Minimum response time of DELETE statements (unit: μs).
avg_delete_elapse	bigint	Average response time of DELETE statements (unit: μs).

13.2.6.7 SUMMARY_WORKLOAD_SQL_ELAPSE_TIME

SUMMARY_WORKLOAD_SQL_ELAPSE_TIME collects SELECT, UPDATE, INSERT, and DELETE (SUID) statistics on workloads (services) of the primary database node, as shown in **Table 13-77**.

Table 13-77 SUMMARY_WORKLOAD_SQL_ELAPSE_TIME columns

Name	Type	Description
node_name	name	Node name.
workload	name	Workload name.
total_select_elapse	bigint	Total response time of SELECT statements (unit: μs).
max_select_elapse	bigint	Maximum response time of SELECT statements (unit: μs).

Name	Type	Description
min_select_elapse	bigint	Minimum response time of SELECT statements (unit: μs).
avg_select_elapse	bigint	Average response time of SELECT statements (unit: μs).
total_update_elapse	bigint	Total response time of UPDATE statements (unit: μs).
max_update_elapse	bigint	Maximum response time of UPDATE statements (unit: μs).
min_update_elapse	bigint	Minimum response time of UPDATE statements (unit: μs).
avg_update_elapse	bigint	Average response time of UPDATE statements (unit: μs).
total_insert_elapse	bigint	Total response time of INSERT statements (unit: μs).
max_insert_elapse	bigint	Maximum response time of INSERT statements (unit: μs).
min_insert_elapse	bigint	Minimum response time of INSERT statements (unit: μs).
avg_insert_elapse	bigint	Average response time of INSERT statements (unit: μs).
total_delete_elapse	bigint	Total response time of DELETE statements (unit: μs).
max_delete_elapse	bigint	Maximum response time of DELETE statements (unit: μs).
min_delete_elapse	bigint	Minimum response time of DELETE statements (unit: μs).
avg_delete_elapse	bigint	Average response time of DELETE statements (unit: μs).

13.2.6.8 USER_TRANSACTION

USER_TRANSACTION collects statistics about transactions executed by users. Users with the monadmin permission can view information about transactions executed by all users, as shown in **Table 13-78**.

Table 13-78 USER_TRANSACTION columns

Name	Type	Description
username	name	Username.

Name	Type	Description
commit_counter	bigint	Number of user transactions committed.
rollback_counter	bigint	Number of user transactions rolled back.
resp_min	bigint	Minimum response time of user transactions (unit: μs).
resp_max	bigint	Maximum response time of user transactions (unit: μs).
resp_avg	bigint	Average response time of user transactions (unit: μs).
resp_total	bigint	Total response time of user transactions (unit: μs).
bg_commit_counter	bigint	Number of background transactions committed.
bg_rollback_counter	bigint	Number of background transactions rolled back.
bg_resp_min	bigint	Minimum response time of background transactions (unit: μs).
bg_resp_max	bigint	Maximum response time of background transactions (unit: μs).
bg_resp_avg	bigint	Average response time of background transactions (unit: μs).
bg_resp_total	bigint	Total response time of background transactions (unit: μs).

13.2.6.9 GLOBAL_USER_TRANSACTION

GLOBAL_USER_TRANSACTION collects statistics about transactions executed by all users, as shown in **Table 13-79**.

Table 13-79 GLOBAL_USER_TRANSACTION columns

Name	Type	Description
node_name	name	Node name.
username	name	Username.
commit_counter	bigint	Number of user transactions committed.
rollback_counter	bigint	Number of user transactions rolled back.

Name	Type	Description
resp_min	bigint	Minimum response time of user transactions (unit: μs).
resp_max	bigint	Maximum response time of user transactions (unit: μs).
resp_avg	bigint	Average response time of user transactions (unit: μs).
resp_total	bigint	Total response time of user transactions (unit: μs).
bg_commit_counter	bigint	Number of background transactions committed.
bg_rollback_counter	bigint	Number of background transactions rolled back.
bg_resp_min	bigint	Minimum response time of background transactions (unit: μs).
bg_resp_max	bigint	Maximum response time of background transactions (unit: μs).
bg_resp_avg	bigint	Average response time of background transactions (unit: μs).
bg_resp_total	bigint	Total response time of background transactions (unit: μs).

13.2.7 Session/Thread

13.2.7.1 SESSION_STAT

Collects statistics about session status on the current node based on session threads or the **AutoVacuum** thread, as shown in **Table 13-80**.

Table 13-80 SESSION_STAT columns

Name	Type	Description
sessid	text	Thread start time and ID
statid	integer	Statistics ID
statname	text	Name of the statistics session
statunit	text	Unit of the statistics session
value	bigint	Value of the statistics session

13.2.7.2 GLOBAL_SESSION_STAT

Collects statistics about session status on each node based on session threads or the AutoVacuum thread, as shown in **Table 13-81**.

Table 13-81 GLOBAL_SESSION_STAT columns

Name	Type	Description
node_name	name	Node name.
sessid	text	Thread start time and ID.
statid	integer	Statistics ID.
statname	text	Name of the statistics session.
statunit	text	Unit of the statistics session.
value	bigint	Value of the statistics session.

13.2.7.3 SESSION_TIME

Collects statistics about the running time of session threads and time consumed in each execution phase on the current node, as shown in **Table 13-82**.

Table 13-82 SESSION_TIME columns

Name	Type	Description
sessid	text	Thread start time and ID.
stat_id	integer	Statistics ID.
stat_name	text	Session type.
value	bigint	Session value.

13.2.7.4 GLOBAL_SESSION_TIME

Collects statistics about the running time of session threads and time consumed in each execution phase on each node, as shown in **Table 13-83**.

Table 13-83 GLOBAL_SESSION_TIME columns

Name	Type	Description
node_name	name	Node name.
sessid	text	Thread start time and ID.
stat_id	integer	Statistics ID.

Name	Type	Description
stat_name	text	Session type.
value	bigint	Session value.

13.2.7.5 SESSION_MEMORY

Collects statistics about memory usage at the session level in the unit of MB, including all the memory allocated to GaussDB and stream threads on DNs for jobs currently executed by users, as shown in **Table 13-84**.

Table 13-84 SESSION_MEMORY columns

Name	Type	Description
sessid	text	Thread start time and ID.
init_mem	integer	Memory allocated to the currently executed job before the job enters the executor.
used_mem	integer	Memory allocated to the currently executed job.
peak_mem	integer	Peak memory allocated to the currently executed job.

13.2.7.6 GLOBAL_SESSION_MEMORY

Collects statistics about memory usage at the session level on each node in the unit of MB, including all the memory allocated to GaussDB and stream threads on DNs for jobs currently executed by users, as shown in **Table 13-85**.

Table 13-85 GLOBAL_SESSION_MEMORY columns

Name	Type	Description
node_name	name	Node name.
sessid	text	Thread start time and ID.
init_mem	integer	Memory allocated to the currently executed job before the job enters the executor.
used_mem	integer	Memory allocated to the currently executed job.
peak_mem	integer	Peak memory allocated to the currently executed job.

13.2.7.7 SESSION_MEMORY_DETAIL

Collects statistics on the memory usage of threads on the current node, in the unit specified by **MemoryContext**, as shown in **Table 13-86**.

Table 13-86 SESSION_MEMORY_DETAIL columns

Name	Type	Description
sessid	text	Thread start time and ID.
sesstype	text	Thread name.
contextname	text	Name of the memory context.
level	smallint	Level of memory context importance.
parent	text	Name of the parent memory context.
totalsize	bigint	Size of the allocated memory (unit: byte).
freesize	bigint	Size of the idle memory (unit: byte).
usedsize	bigint	Size of the used memory (unit: byte).

13.2.7.8 GLOBAL_SESSION_MEMORY_DETAIL

Collects statistics about thread memory usage on each node by **MemoryContext**, as shown in **Table 13-87**.

Table 13-87 GLOBAL_SESSION_MEMORY_DETAIL columns

Name	Type	Description
node_name	name	Node name.
sessid	text	Thread start time and ID.
sesstype	text	Thread name.
contextname	text	Name of the memory context.
level	smallint	Level of memory context importance.
parent	text	Name of the parent memory context.
totalsize	bigint	Size of the allocated memory (unit: byte).
freesize	bigint	Size of the idle memory (unit: byte).
usedsize	bigint	Size of the used memory (unit: byte).

13.2.7.9 SESSION_STAT_ACTIVITY

Displays information about threads that are running on the current node, as described in **Table 13-88**.

Table 13-88 SESSION_STAT_ACTIVITY columns

Name	Type	Description
datid	oid	OID of the database that the user session connects to in the backend.
datname	name	Name of the database that the user session connects to in the backend.
pid	bigint	ID of the backend thread.
usesysid	oid	OID of the user logged in to the backend.
usename	name	Name of the user logged in to the backend.
application_name	text	Name of the application connected to the backend.
client_addr	inet	IP address of the client connected to the backend. If this column is **NULL**, it indicates either the client is connected via a Unix socket on the server or this is an internal thread, for example, **autovacuum** is displayed for the thread autovacuum.
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of **client_addr**. This column will be non-null only for IP connections and only when **log_hostname** is enabled.
client_port	integer	TCP port number that the client uses for communication with the backend (**–1** if a Unix socket is used).
backend_start	timestamp with time zone	Time when this process was started, that is, when the client connected to the server.
xact_start	timestamp with time zone	Time when current transaction was started (null if no transaction is active). If the current query is the first of its transaction, the value of this column is the same as that of the **query_start** column.

Name	Type	Description
query_start	timestamp with time zone	Time when the currently active query was started, or time when the last query was started if **state** is not **active**. For a stored procedure, function, or package, the first query time is queried and does not change with the running of statements in the stored procedure.
state_change	timestamp with time zone	Time when **state** was last modified.
waiting	Boolean	Specifies whether the backend is currently waiting for a lock. If yes, the value is **true**.
enqueue	text	Unsupported currently.

Name	Type	Description
state	text	Overall status of this backend. The value can be: • **active**: The backend is executing a query. • **idle**: The backend is waiting for a new client command. • **idle in transaction**: The backend is in a transaction, but is not currently executing a query. • **idle in transaction (aborted)**: The backend is in a transaction, but there are statements failed in the transaction. • **fastpath function call**: The backend is executing a fast-path function. • **disabled**: This state is reported if **track_activities** is disabled in this backend. **NOTE** Common users can view their own session status only. The state information of other accounts is empty. For example, after the **judy** user is connected to the database, the state information of the **joe** user and the initial user **omm** in **pg_stat_activity** is empty. gaussdb=# SELECT datname, usename, usesysid,state,pid FROM pg_stat_activity; datname | usename | usesysid | state | pid ----------+---------+----------+-------- +-----------------testdb | omm | 10 | | 139968752121616 testdb | omm | 10 | | 139968903116560 db_tpcds | judy | 16398 | active | 139968391403280 testdb | omm | 10 | | 139968643069712 testdb | omm | 10 | | 139968680818448 testdb | joe | 16390 | | 139968563377936 (6 rows)
resource_pool	name	Resource pool used by the user.
query_id	bigint	ID of a query.
query	text	Text of this backend's latest query. If the value of **state** is **active**, this column shows the ongoing query. In all other states, it shows the last query that was executed.
unique_sql_id	bigint	Unique SQL statement ID.

Name	Type	Description
trace_id	text	Trace ID passed by the driver, which is used to identify a request of an application.

13.2.7.10 GLOBAL_SESSION_STAT_ACTIVITY

Displays information about threads that are running on each node in the database, as described in **Table 13-89**.

Table 13-89 GLOBAL_SESSION_STAT_ACTIVITY columns

Name	Type	Description
coorname	text	Database process name.
datid	oid	OID of the database that the user session connects to in the backend.
datname	text	Name of the database that the user session connects to in the backend.
pid	bigint	ID of the backend thread.
usesysid	oid	OID of the user logged in to the backend.
usename	text	Name of the user logged in to the backend.
application_name	text	Name of the application connected to the backend.
client_addr	inet	IP address of the client connected to the backend. If this column is **NULL**, it indicates either the client is connected via a Unix socket on the server or this is an internal thread, for example, **autovacuum** is displayed for the thread autovacuum.
client_hostname	text	Host name of the connected client, as reported by a reverse DNS lookup of **client_addr**. This column will be non-null only for IP connections and only when **log_hostname** is enabled.
client_port	integer	TCP port number that the client uses for communication with the backend (**–1** if a Unix socket is used).
backend_start	timestamp with time zone	Time when this process was started, that is, when the client connected to the server.

Name	Type	Description
xact_start	timestamp with time zone	Time when current transaction was started (null if no transaction is active). If the current query is the first of its transaction, the value of this column is the same as that of the **query_start** column.
query_start	timestamp with time zone	Time when the currently active query was started, or time when the last query was started if **state** is not **active**. For a stored procedure, function, or package, the first query time is queried and does not change with the running of statements in the stored procedure.
state_change	timestamp with time zone	Time when **state** was last modified.
waiting	Boolean	Specifies whether the backend is currently waiting for a lock. If yes, the value is **true**.
enqueue	text	Unsupported currently.

Name	Type	Description
state	text	Overall status of this backend. The value can be: • **active**: The backend is executing a query. • **idle**: The backend is waiting for a new client command. • **idle in transaction**: The backend is in a transaction, but is not currently executing a query. • **idle in transaction (aborted)**: The backend is in a transaction, but there are statements failed in the transaction. • **fastpath function call**: The backend is executing a fast-path function. • **disabled**: This state is reported if **track_activities** is disabled in this backend. **NOTE** Common users can view their own session status only. The state information of other accounts is empty. For example, after the **judy** user is connected to the database, the state information of the **joe** user and the initial user **omm** in **pg_stat_activity** is empty. gaussdb=# SELECT datname, username, usesysid,state,pid FROM pg_stat_activity; datname | username | usesysid | state | pid ----------+----------+----------+-------- +-----------------testdb | omm | 10 | | 139968752121616 testdb | omm | 10 | | 139968903116560 db_tpcds | judy | 16398 | active | 139968391403280 testdb | omm | 10 | | 139968643069712 testdb | omm | 10 | | 139968680818448 testdb | joe | 16390 | | 139968563377936 (6 rows)
resource_pool	name	Resource pool used by the user.
query_id	bigint	ID of a query.
query	text	Text of this backend's latest query. If the value of **state** is **active**, this column shows the ongoing query. In all other states, it shows the last query that was executed.
unique_sql_id	bigint	Unique SQL statement ID.

Name	Type	Description
trace_id	text	Trace ID passed by the driver, which is used to identify a request of an application.

13.2.7.11 THREAD_WAIT_STATUS

In this view, you can check the blocking and waiting status of the backend threads and auxiliary threads on the current node, as shown in **Table 13-90**. For details about events, see **Table 12-412**, **Table 12-413**, **Table 12-414**, and **Table 12-415**.

Table 13-90 THREAD_WAIT_STATUS columns

Name	Type	Description
node_name	text	Node name.
db_name	text	Database name.
thread_name	text	Thread name.
query_id	bigint	Query ID. The value of this column is the same as that of **debug_query_id**.
tid	bigint	Thread ID of the current thread.
sessionid	bigint	Session ID.
lwtid	integer	Lightweight thread ID of the current thread.
psessionid	bigint	Parent thread of the streaming thread.
tlevel	integer	Level of the streaming thread.
smpid	integer	Concurrent thread ID.
wait_status	text	Waiting status of the current thread. For details about the waiting status, see **Table 12-412**.
wait_event	text	If **wait_status** is **acquire lock**, **acquire lwlock**, or **wait io**, this column describes the lock, lightweight lock, or I/O information. Otherwise, this column is empty.
locktag	text	Information about the lock that the current thread is waiting for.
lockmode	text	Lock mode that the current thread is waiting to obtain. The values include modes in the table-level lock, row-level lock, and page-level lock.
block_sessionid	bigint	ID of the session that blocks the current thread from obtaining the lock.

Name	Type	Description
global_sessionid	text	Global session ID.

13.2.7.12 GLOBAL_THREAD_WAIT_STATUS

In this view, you can check the blocking and waiting status of the backend threads and auxiliary threads on all nodes, as shown in **Table 13-91**. For details about events, see **Table 12-412**, **Table 12-413**, **Table 12-414**, and **Table 12-415**.

In GLOBAL_THREAD_WAIT_STATUS, you can see all the call hierarchy relationships between threads of the SQL statements on all nodes in the database, and the block waiting status for each thread. With this view, you can easily locate the causes of process hang and similar issues.

The definitions of GLOBAL_THREAD_WAIT_STATUS and THREAD_WAIT_STATUS are the same, because the GLOBAL_THREAD_WAIT_STATUS view is essentially the query summary of the THREAD_WAIT_STATUS view on each node in the database.

Table 13-91 GLOBAL_THREAD_WAIT_STATUS columns

Name	Type	Description
node_name	text	Node name.
db_name	text	Database name.
thread_name	text	Thread name.
query_id	bigint	Query ID. The value of this column is the same as that of **debug_query_id**.
tid	bigint	Thread ID of the current thread.
sessionid	bigint	Session ID.
lwtid	integer	Lightweight thread ID of the current thread.
psessionid	bigint	Parent thread of the streaming thread.
tlevel	integer	Level of the streaming thread.
smpid	integer	Concurrent thread ID.
wait_status	text	Waiting status of the current thread. For details about the waiting status, see **Table 12-412**.
wait_event	text	If **wait_status** is **acquire lock**, **acquire lwlock**, or **wait io**, this column describes the lock, lightweight lock, or I/O information. Otherwise, this column is empty.

Name	Type	Description
locktag	text	Information about the lock that the current thread is waiting for.
lockmode	text	Lock mode that the current thread is waiting to obtain. The values include modes in the table-level lock, row-level lock, and page-level lock.
block_sessionid	bigint	ID of the session that blocks the current thread from obtaining the lock.
global_sessionid	text	Global session ID.

13.2.7.13 LOCAL_THREADPOOL_STATUS

LOCAL_THREADPOOL_STATUS displays the status of worker threads and sessions in a thread pool, as shown in **Table 13-92**. This view is valid only when **enable_thread_pool** is set to **on**.

Table 13-92 LOCAL_THREADPOOL_STATUS columns

Name	Type	Description
node_name	text	Database process name.
group_id	integer	ID of the thread pool group.
bind_numa_id	integer	NUMA ID to which the thread pool group is bound.
bind_cpu_number	integer	Information about the CPU to which the thread pool group is bound. If no CPUs are bound, the value is **NULL**.
listener	integer	Number of listener threads in the thread pool group.

Name	Type	Description
worker_info	text	Information about threads in the thread pool, including: • **default**: Number of initial threads in the thread pool group • **new**: Number of new threads in the thread pool group • **expect**: Expected number of threads in the thread pool group • **actual**: Actual number of threads in the thread pool group • **idle**: Number of idle threads in the thread pool group • **pending**: Number of pending threads in the thread pool group
session_info	text	Information about sessions in the thread pool, including: • **total**: Total number of sessions in the thread pool group • **waiting**: Number of sessions pending scheduling in the thread pool group • **running**: Number of running sessions in the thread pool group • **idle**: Number of idle sessions in the thread pool group
stream_info	text	Stream pool information, including: • **total**: total number of threads in the stream pool group. • **running**: number of threads that are being executed in the stream pool. • **idle**: number of idle threads in the stream pool.

13.2.7.14 GLOBAL_THREADPOOL_STATUS

GLOBAL_THREADPOOL_STATUS displays the status of worker threads and sessions in thread pools on all nodes. For details, see **Table 13-92**.

13.2.7.15 LOCAL_ACTIVE_SESSION

LOCAL_ACTIVE_SESSION displays samples in the ACTIVE SESSION PROFILE memory on the current node, as shown in **Table 13-93**.

Table 13-93 LOCAL_ACTIVE_SESSION columns

Name	Type	Description
sampleid	bigint	Sample ID
sample_time	timestamp with time zone	Sampling time
need_flush_sample	Boolean	Specifies whether the sample needs to be flushed to disks.
databaseid	oid	Database ID
thread_id	bigint	Thread ID
sessionid	bigint	Session ID
start_time	timestamp with time zone	Start time of a session
event	text	Event name
lwtid	integer	Lightweight thread ID of the current thread.
psessionid	bigint	Parent thread of the streaming thread.
tlevel	integer	Level of the streaming thread, which corresponds to the level (**id**) of the execution plan
smpid	integer	Concurrent thread ID in SMP execution mode
userid	oid	ID of a session user
application_name	text	Name of an application
client_addr	inet	IP address of a client
client_hostname	text	Name of a client
client_port	integer	TCP port number used by a client to communicate with the backend
query_id	bigint	Debug query ID.
unique_query_id	bigint	Unique query ID.
user_id	oid	User ID in the key of the unique query

Name	Type	Description
cn_id	integer	CN ID. On a DN, this parameter indicates the ID of the node that delivers the unique SQL statement, that is, the value of **cn_id** in the key of the unique query.
unique_query	text	Standardized text string of the unique SQL statement
locktag	text	Information of a lock that the session waits for, which can be parsed using **locktag_decode**
lockmode	text	Mode of a lock that the session waits for
block_sessionid	bigint	Blocks a session from obtaining the session ID of a lock if the session is waiting for the lock.
final_block_sessionid	bigint	ID of the blocked session at the source end
wait_status	text	Provides more details about an event column.
global_sessionid	text	Global session ID
xact_start_time	timestamp with time zone	Start time of the transaction
query_start_time	timestamp with time zone	Time when the statement starts to be executed
state	text	Current statement state The value can be **active**, **idle in transaction**, **fastpath function call**, **idle in transaction (aborted)**, **disabled**, or **retrying**.
event_start_time	timestamp with time zone	Start time of a wait event.
current_xid	xid	Current transaction ID.
top_xid	xid	Top-level transaction ID.

13.2.8 Transaction

13.2.8.1 TRANSACTIONS_PREPARED_XACTS

Displays information about transactions that are currently prepared for two-phase commit on the current node, as described in **Table 13-94**.

Table 13-94 TRANSACTIONS_PREPARED_XACTS columns

Name	Type	Description
transaction	xid	Numeric transaction identifier of the prepared transaction
gid	text	Global transaction identifier that was assigned to the transaction
prepared	timestamp with time zone	Time at which the transaction is prepared for commit
owner	name	Name of the user who executes the transaction
database	name	Name of the database in which the transaction is executed

13.2.8.2 SUMMARY_TRANSACTIONS_PREPARED_XACTS

Displays information about transactions that are currently prepared for two-phase commit on the primary database node, as described in **Table 13-95**.

Table 13-95 SUMMARY_TRANSACTIONS_PREPARED_XACTS columns

Name	Type	Description
transaction	xid	Numeric transaction identifier of the prepared transaction
gid	text	Global transaction identifier that was assigned to the transaction
prepared	timestamp with time zone	Time at which the transaction is prepared for commit
owner	name	Name of the user who executes the transaction
database	name	Name of the database in which the transaction is executed

13.2.8.3 GLOBAL_TRANSACTIONS_PREPARED_XACTS

Displays information about transactions that are currently prepared for two-phase commit on the primary database node, as described in **Table 13-96**.

Table 13-96 GLOBAL_TRANSACTIONS_PREPARED_XACTS columns

Name	Type	Description
transaction	xid	Numeric transaction identifier of the prepared transaction
gid	text	Global transaction identifier that was assigned to the transaction
prepared	timestamp with time zone	Time at which the transaction is prepared for commit
owner	name	Name of the user who executes the transaction
database	name	Name of the database in which the transaction is executed

13.2.8.4 TRANSACTIONS_RUNNING_XACTS

Displays information about running transactions on the current node, as described in **Table 13-97**.

Table 13-97 TRANSACTIONS_RUNNING_XACTS columns

Name	Type	Description
handle	integer	Slot handle in the transaction manager corresponding to a transaction. The value is fixed at **–1**.
gxid	xid	Transaction ID
state	tinyint	Transaction status (**3**: prepared or **0**: starting)
node	text	Node name.
xmin	xid	Minimum transaction ID on the node
vacuum	boolean	Specifies whether the current transaction is a lazy vacuum (VACUUM only if necessary) transaction. • **true**: yes • **false**: no
timeline	bigint	Number of database restarts
prepare_xid	xid	Transaction ID in the **prepared** state. If the state is not **prepared**, the value is **0**.
pid	bigint	Thread ID corresponding to the transaction
next_xid	xid	Transaction ID sent by other nodes to the current node. The value is fixed at **0**.

13.2.8.5 SUMMARY_TRANSACTIONS_RUNNING_XACTS

Displays the summary of running transactions on the primary database node, as described in **Table 13-98**.

Table 13-98 SUMMARY_TRANSACTIONS_RUNNING_XACTS columns

Name	Type	Description
handle	integer	Slot handle in the transaction manager corresponding to a transaction. The value is fixed at **–1**.
gxid	xid	Transaction ID.
state	tinyint	Transaction status (**3**: prepared; or **0**: starting).
node	text	Node name.
xmin	xid	Minimum transaction ID on the node.
vacuum	boolean	Specifies whether the current transaction is a lazy vacuum (VACUUM only if necessary) transaction. • **true**: yes • **false**: no
timeline	bigint	Number of database restarts.
prepare_xid	xid	Transaction ID in the **prepared** state. If the status is not **prepared**, the value is **0**.
pid	bigint	Thread ID corresponding to the transaction.
next_xid	xid	Transaction ID sent by other nodes to the current node. The value is fixed at **0**.

13.2.8.6 GLOBAL_TRANSACTIONS_RUNNING_XACTS

Displays the summary of running transactions on the primary database node, as described in **Table 13-99**.

Table 13-99 GLOBAL_TRANSACTIONS_RUNNING_XACTS columns

Name	Type	Description
handle	integer	Slot handle in the transaction manager corresponding to a transaction. The value is fixed at **–1**.
gxid	xid	Transaction ID.
state	tinyint	Transaction status (**3**: prepared; or **0**: starting).
node	text	Node name.

Name	Type	Description
xmin	xid	Minimum transaction ID on the node.
vacuum	boolean	Specifies whether the current transaction is a lazy vacuum (VACUUM only if necessary) transaction. • **true**: yes • **false**: no
timeline	bigint	Number of database restarts.
prepare_xid	xid	Transaction ID in the **prepared** state. If the status is not **prepared**, the value is **0**.
pid	bigint	Thread ID corresponding to the transaction.
next_xid	xid	Transaction ID sent by other nodes to the current node. The value is fixed at **0**.

13.2.9 Query

13.2.9.1 STATEMENT

Obtains information about executed statements (normalized SQL statements) on the current node, as described in **Table 13-100**. You can view all statistics about normalized SQL statements received by the primary database node and other database nodes, whereas you can view only the statistics about normalized SQL statements executed on other database nodes.

NOTE

The **unique_sql_id** generated by different **savepoint_name** values is different. When a large number of **savepoint_name** is used, the number of **unique_sql_id** values generated in the system increases rapidly. If the number of **unique_sql_id** values is greater than the number of **instr_unique_sql_count** values, the newly generated **unique_sql_id** information is not counted.

Table 13-100 STATEMENT columns

Name	Type	Description
node_name	name	Database process name.
node_id	integer	Node ID.
user_name	name	Username.
user_id	oid	OID of the user.
unique_sql_id	bigint	ID of the normalized SQL statement.

Name	Type	Description
query	text	Normalized SQL statement. Note: The length is controlled by **track_activity_query_size**.
n_calls	bigint	Number of calls.
min_elapse_time	bigint	Minimum execution time of the SQL statement in the kernel (unit: μs).
max_elapse_time	bigint	Maximum execution time of the SQL statement in the kernel (unit: μs).
total_elapse_time	bigint	Total execution time of the SQL statement in the kernel (unit: μs).
n_returned_rows	bigint	Number of rows in the result set returned by the **SELECT** statement.
n_tuples_fetched	bigint	Number of rows randomly scanned.
n_tuples_returned	bigint	Number of rows sequentially scanned.
n_tuples_inserted	bigint	Number of rows inserted.
n_tuples_updated	bigint	Number of rows updated.
n_tuples_deleted	bigint	Number of rows deleted.
n_blocks_fetched	bigint	Number of buffer block access times.
n_blocks_hit	bigint	Number of buffer block hits.
n_soft_parse	bigint	Number of soft parsing times. The value of **n_soft_parse** plus the value of **n_hard_parse** may be greater than the value of **n_calls** because the number of subqueries is not counted in the value of **n_calls**.
n_hard_parse	bigint	Number of hard parsing times. The value of **n_soft_parse** plus the value of **n_hard_parse** may be greater than the value of **n_calls** because the number of subqueries is not counted in the value of **n_calls**.
db_time	bigint	Valid DB time, which is accumulated if multiple threads are involved (unit: μs).
cpu_time	bigint	CPU time (unit: μs).
execution_time	bigint	Execution time in the executor (unit: μs).
parse_time	bigint	SQL parsing time (unit: μs).
plan_time	bigint	SQL plan generation time (unit: μs).

Name	Type	Description
rewrite_time	bigint	SQL rewriting time (unit: μs).
pl_execution_time	bigint	Execution time of PL/pgSQL (unit: μs).
pl_compilation_time	bigint	Compilation time of PL/pgSQL (unit: μs).
data_io_time	bigint	I/O time (unit: μs).
net_send_info	text	Network status of messages sent through a physical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This column is not supported in standalone mode.
net_recv_info	text	Network status of messages received through a physical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This column is not supported in standalone mode.
net_stream_send_info	text	Network status of messages sent through a logical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This column is not supported in standalone mode.
net_stream_recv_info	text	Network status of messages received through a logical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This column is not supported in standalone mode.
last_updated	timestamp with time zone	Last time when the statement was updated.
sort_count	bigint	Sorting count
sort_time	bigint	Sorting duration (unit: μs).
sort_mem_used	bigint	Size of work memory used during sorting (unit: KB).
sort_spill_count	bigint	Count of file writing when data is flushed to disks during sorting.
sort_spill_size	bigint	File size used when data is flushed to disks during sorting (unit: KB).
hash_count	bigint	Hashing count.
hash_time	bigint	Hashing duration (unit: μs).

Name	Type	Description
hash_mem_used	bigint	Size of work memory used during hashing (unit: KB).
hash_spill_count	bigint	Count of file writing when data is flushed to disks during hashing.
hash_spill_size	bigint	File size used when data is flushed to disks during hashing (unit: KB).
parent_unique_sql_id	bigint	Unique ID of the parent SQL statement. The value is **0** for a non-stored procedure substatement.
lock_wait_time	bigint	SQL lock waiting time (unit: μs).

NOTE

n_calls indicates the actual number of calling times. For the FETCH statement in the stored procedure, the actual number of triggering times of the FETCH statement is the increase times of **n_calls** of the statement actually executed by the cursor.

13.2.9.2 SUMMARY_STATEMENT

Obtains all information about executed statements (normalized SQL statements) on the primary database node and other database nodes, as described in **Table 13-101**.

Table 13-101 SUMMARY_STATEMENT columns

Name	Type	Description
node_name	name	Database process name.
node_id	integer	Node ID.
user_name	name	Username.
user_id	oid	OID of the user.
unique_sql_id	bigint	ID of the normalized SQL statement.
query	text	Normalized SQL statement. Note: The length is controlled by **track_activity_query_size**.
n_calls	bigint	Number of calls.
min_elapse_time	bigint	Minimum execution time of the SQL statement in the kernel (unit: μs).
max_elapse_time	bigint	Maximum execution time of the SQL statement in the kernel (unit: μs).

Name	Type	Description
total_elapse_time	bigint	Total execution time of the SQL statement in the kernel (unit: μs).
n_returned_rows	bigint	Number of rows in the result set returned by the **SELECT** statement.
n_tuples_fetched	bigint	Number of rows randomly scanned.
n_tuples_returned	bigint	Number of rows sequentially scanned.
n_tuples_inserted	bigint	Number of rows inserted.
n_tuples_updated	bigint	Number of rows updated.
n_tuples_deleted	bigint	Number of rows deleted.
n_blocks_fetched	bigint	Number of buffer block access times.
n_blocks_hit	bigint	Number of buffer block hits.
n_soft_parse	bigint	Number of soft parsing times.
n_hard_parse	bigint	Number of hard parsing times.
db_time	bigint	Valid DB time, which is accumulated if multiple threads are involved (unit: μs).
cpu_time	bigint	CPU time (unit: μs).
execution_time	bigint	Execution time in the executor (unit: μs).
parse_time	bigint	SQL parsing time (unit: μs).
plan_time	bigint	SQL plan generation time (unit: μs).
rewrite_time	bigint	SQL rewriting time (unit: μs).
pl_execution_time	bigint	Execution time of PL/pgSQL (unit: μs).
pl_compilation_time	bigint	Compilation time of PL/pgSQL (unit: μs).
data_io_time	bigint	I/O time (unit: μs).
net_send_info	text	Network status of messages sent through a physical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This column is not supported in standalone mode.
net_recv_info	text	Network status of messages received through a physical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This column is not supported in standalone mode.

Name	Type	Description
net_stream_send_info	text	Network status of messages sent through a logical connection, including the time (unit: µs), number of calls, and throughput (unit: byte). This column is not supported in standalone mode.
net_stream_recv_info	text	Network status of messages received through a logical connection, including the time (unit: µs), number of calls, and throughput (unit: byte). This column is not supported in standalone mode.
last_updated	timestamp with time zone	Last time when the statement was updated.
sort_count	bigint	Sorting count.
sort_time	bigint	Sorting duration (unit: µs).
sort_mem_used	bigint	Size of work memory used during sorting (unit: KB).
sort_spill_count	bigint	Count of file writing when data is flushed to disks during sorting.
sort_spill_size	bigint	File size used when data is flushed to disks during sorting (unit: KB).
hash_count	bigint	Hashing count.
hash_time	bigint	Hashing duration (unit: µs).
hash_mem_used	bigint	Size of work memory used during hashing (unit: KB).
hash_spill_count	bigint	Count of file writing when data is flushed to disks during hashing.
hash_spill_size	bigint	File size used when data is flushed to disks during hashing (unit: KB).
parent_unique_sql_id	bigint	Unique ID of the parent SQL statement. The value is **0** for a non-stored procedure substatement.
lock_wait_time	bigint	SQL lock waiting time (unit: µs).

13.2.9.3 STATEMENT_COUNT

Displays statistics about five types of running statements (SELECT, INSERT, UPDATE, DELETE, and MERGE INTO) as well as DDL, DML, and DCL statements on the current node of the database, as described in **Table 13-102**.

NOTE

An administrator can query the STATEMENT_COUNT view to view the statistics of all users on the current node. When the database or the node is restarted, the statistics are cleared and the counting restarts. The system counts when a node receives a query, including queries inside the database. For example, when the primary database node receives a query, it is counted on the database node though multiple queries are delivered the database node.

Table 13-102 STATEMENT_COUNT columns

Name	Type	Description
node_name	text	Database process name.
user_name	text	Username.
select_count	bigint	Statistical result of the SELECT statements.
update_count	bigint	Statistical result of the UPDATE statements.
insert_count	bigint	Statistical result of the INSERT statements.
delete_count	bigint	Statistical result of the DELETE statements.
mergeinto_count	bigint	Statistical result of the MERGE INTO statements.
ddl_count	bigint	Number of DDL statements.
dml_count	bigint	Number of DML statements.
dcl_count	bigint	Number of DCL statements.
total_select_elapse	bigint	Total response time of SELECT statements (unit: μs).
avg_select_elapse	bigint	Average response time of SELECT statements (unit: μs).
max_select_elapse	bigint	Maximum response time of SELECT statements (unit: μs).
min_select_elapse	bigint	Minimum response time of SELECT statements (unit: μs).
total_update_elapse	bigint	Total response time of UPDATE statements (unit: μs).
avg_update_elapse	bigint	Average response time of UPDATE statements (unit: μs).
max_update_elapse	bigint	Maximum response time of UPDATE statements (unit: μs).
min_update_elapse	bigint	Minimum response time of UPDATE statements (unit: μs).

Name	Type	Description
total_insert_elapse	bigint	Total response time of INSERT statements (unit: μs).
avg_insert_elapse	bigint	Average response time of INSERT statements (unit: μs).
max_insert_elapse	bigint	Maximum response time of INSERT statements (unit: μs).
min_insert_elapse	bigint	Minimum response time of INSERT statements (unit: μs).
total_delete_elapse	bigint	Total response time of DELETE statements (unit: μs).
avg_delete_elapse	bigint	Average response time of DELETE statements (unit: μs).
max_delete_elapse	bigint	Maximum response time of DELETE statements (unit: μs).
min_delete_elapse	bigint	Minimum response time of DELETE statements (unit: μs).

13.2.9.4 GLOBAL_STATEMENT_COUNT

Displays statistics about five types of running statements (**SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO**) as well as DDL, DML, and DCL statements on each node of the database, as described in **Table 13-103**.

Table 13-103 GLOBAL_STATEMENT_COUNT columns

Name	Type	Description
node_name	text	Node name
user_name	text	Username
select_count	bigint	Statistical result of the **SELECT** statement
update_count	bigint	Statistical result of the **UPDATE** statement
insert_count	bigint	Statistical result of the **INSERT** statement
delete_count	bigint	Statistical result of the **DELETE** statement
mergeinto_count	bigint	Statistical result of the **MERGE INTO** statement
ddl_count	bigint	Number of DDL statements
dml_count	bigint	Number of DML statements

Name	Type	Description
dcl_count	bigint	Number of DCL statements
total_select_elapse	bigint	Total response time of **SELECT** statements (unit: μs)
avg_select_elapse	bigint	Average response time of **SELECT** statements (unit: μs)
max_select_elapse	bigint	Maximum response time of **SELECT** statements (unit: μs)
min_select_elapse	bigint	Minimum response time of **SELECT** statements (unit: μs)
total_update_elapse	bigint	Total response time of **UPDATE** statements (unit: μs)
avg_update_elapse	bigint	Average response time of **UPDATE** statements (unit: μs)
max_update_elapse	bigint	Maximum response time of **UPDATE** statements (unit: μs)
min_update_elapse	bigint	Minimum response time of **UPDATE** statements (unit: μs)
total_insert_elapse	bigint	Total response time of **INSERT** statements (unit: μs)
avg_insert_elapse	bigint	Average response time of **INSERT** statements (unit: μs)
max_insert_elapse	bigint	Maximum response time of **INSERT** statements (unit: μs)
min_insert_elapse	bigint	Minimum response time of **INSERT** statements (unit: μs)
total_delete_elapse	bigint	Total response time of **DELETE** statements (unit: μs)
avg_delete_elapse	bigint	Average response time of **DELETE** statements (unit: μs)
max_delete_elapse	bigint	Maximum response time of **DELETE** statements (unit: μs)
min_delete_elapse	bigint	Minimum response time of **DELETE** statements (unit: μs)

13.2.9.5 SUMMARY_STATEMENT_COUNT

Displays statistics about five types of running statements (**SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO**) as well as DDL, DML, and DCL statements on all nodes (database nodes) of the database, as described in **Table 13-104**.

Table 13-104 SUMMARY_STATEMENT_COUNT columns

Name	Type	Description
user_name	text	Username
select_count	numeric	Statistical result of the **SELECT** statement
update_count	numeric	Statistical result of the **UPDATE** statement
insert_count	numeric	Statistical result of the **INSERT** statement
delete_count	numeric	Statistical result of the **DELETE** statement
mergeinto_count	numeric	Statistical result of the **MERGE INTO** statement
ddl_count	numeric	Number of DDL statements
dml_count	numeric	Number of DML statements
dcl_count	numeric	Number of DCL statements
total_select_elapse	numeric	Total response time of **SELECT** statements (unit: μs)
avg_select_elapse	bigint	Average response time of **SELECT** statements (unit: μs)
max_select_elapse	bigint	Maximum response time of **SELECT** statements (unit: μs)
min_select_elapse	bigint	Minimum response time of **SELECT** statements (unit: μs)
total_update_elapse	numeric	Total response time of **UPDATE** statements (unit: μs)
avg_update_elapse	bigint	Average response time of **UPDATE** statements (unit: μs)
max_update_elapse	bigint	Maximum response time of **UPDATE** statements (unit: μs)
min_update_elapse	bigint	Minimum response time of **UPDATE** statements (unit: μs)
total_insert_elapse	numeric	Total response time of **INSERT** statements (unit: μs)
avg_insert_elapse	bigint	Average response time of **INSERT** statements (unit: μs)
max_insert_elapse	bigint	Maximum response time of **INSERT** statements (unit: μs)

Name	Type	Description
min_insert_elapse	bigint	Minimum response time of **INSERT** statements (unit: μs)
total_delete_elapse	numeric	Total response time of **DELETE** statements (unit: μs)
avg_delete_elapse	bigint	Average response time of **DELETE** statements (unit: μs)
max_delete_elapse	bigint	Maximum response time of **DELETE** statements (unit: μs)
min_delete_elapse	bigint	Minimum response time of **DELETE** statements (unit: μs)

13.2.9.6 STATEMENT_RESPONSETIME_PERCENTILE

Obtains the response times of 80% and 95% SQL statements in the database, as described in **Table 13-105**.

Table 13-105 STATEMENT_RESPONSETIME_PERCENTILE columns

Name	Type	Description
p80	bigint	Response time of 80% SQL statements in the database (unit: μs).
p95	bigint	Response time of 95% SQL statements in the database (unit: μs).

13.2.9.7 STATEMENT_HISTORY

Displays information about statements executed on the current node. The result can be queried only in the system database but cannot be queried in the user database, as described in **Table 13-106**.

Table 13-106 STATEMENT_HISTORY columns

Name	Type	Description
db_name	name	Database name.
schema_name	name	Schema name.
origin_node	integer	Node name.
user_name	name	Username.
application_name	text	Name of the application that sends a request.

Name	Type	Description
client_addr	text	IP address of the client that sends a request.
client_port	integer	Port number of the client that sends a request.
unique_query_id	bigint	ID of the normalized SQL statement.
debug_query_id	bigint	ID of the unique SQL statement.
query	text	Normalized SQL statement.
start_time	timestamp with time zone	Time when a statement starts.
finish_time	timestamp with time zone	Time when a statement ends.
slow_sql_threshold	bigint	Standard for slow SQL statement execution.
transaction_id	bigint	Transaction ID.
thread_id	bigint	ID of an execution thread.
session_id	bigint	Session ID of a user.
n_soft_parse	bigint	Number of soft parsing times. The value of **n_soft_parse** plus the value of **n_hard_parse** may be greater than the value of **n_calls** because the number of subqueries is not counted in the value of **n_calls**.
n_hard_parse	bigint	Number of hard parsing times. The value of **n_soft_parse** plus the value of **n_hard_parse** may be greater than the value of **n_calls** because the number of subqueries is not counted in the value of **n_calls**.
query_plan	text	Statement execution plan.
n_returned_rows	bigint	Number of rows in the result set returned by the **SELECT** statement.
n_tuples_fetched	bigint	Number of rows randomly scanned.
n_tuples_returned	bigint	Number of rows sequentially scanned.
n_tuples_inserted	bigint	Number of rows inserted.

Name	Type	Description
n_tuples_updated	bigint	Number of rows updated.
n_tuples_deleted	bigint	Number of rows deleted.
n_blocks_fetched	bigint	Number of buffer block access times.
n_blocks_hit	bigint	Number of buffer block hits.
db_time	bigint	Valid DB time, which is accumulated if multiple threads are involved (unit: μs).
cpu_time	bigint	CPU time (unit: μs).
execution_time	bigint	Execution time in the executor (unit: μs).
parse_time	bigint	SQL parsing time (unit: μs).
plan_time	bigint	SQL plan generation time (unit: μs).
rewrite_time	bigint	SQL rewriting time (unit: μs).
pl_execution_time	bigint	Execution time of PL/pgSQL (unit: μs).
pl_compilation_time	bigint	Compilation time of PL/pgSQL (unit: μs).
data_io_time	bigint	I/O time (unit: μs).
net_send_info	text	Network status of messages sent through a physical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This can be used to analyze the network overhead of SQL statements in a distributed system and is not supported in standalone system. Example: {"time":xxx, "n_calls":xxx, "size":xxx}.
net_recv_info	text	Network status of messages received through a physical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This can be used to analyze the network overhead of SQL statements in a distributed system and is not supported in standalone system. Example: {"time":xxx, "n_calls":xxx, "size":xxx}.

Name	Type	Description
net_stream_send_info	text	Network status of messages sent through a logical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This can be used to analyze the network overhead of SQL statements in a distributed system and is not supported in standalone system. Example: {"time":xxx, "n_calls":xxx, "size":xxx}.
net_stream_recv_info	text	Network status of messages received through a logical connection, including the time (unit: μs), number of calls, and throughput (unit: byte). This can be used to analyze the network overhead of SQL statements in a distributed system and is not supported in standalone system. Example: {"time":xxx, "n_calls":xxx, "size":xxx}.
lock_count	bigint	Number of locks.
lock_time	bigint	Time required for locking.
lock_wait_count	bigint	Number of lock waits.
lock_wait_time	bigint	Time required for lock waiting.
lock_max_count	bigint	Maximum number of locks.
lwlock_count	bigint	Number of lightweight locks (reserved).
lwlock_wait_count	bigint	Number of lightweight lock waits.
lwlock_time	bigint	Time required for lightweight locking (reserved).
lwlock_wait_time	bigint	Time required for lightweight locking.

Name	Type	Description
details	bytea	List of wait events and statement lock events. When the value of **track_stmt_stat_level** is greater than or equal to **L0**, the list of wait events starts to be recorded. Statistics about the wait events of the current statement are displayed. For details about the events, see **Table 12-412**, **Table 12-413**, **Table 12-414**, and **Table 12-415**. For details about the impact of each transaction lock on services, see **LOCK**. When the value of **track_stmt_stat_level** is **L2**, the list of statement lock events starts to be recorded. The list records events in chronological order. The number of records is determined by the **track_stmt_details_size** parameter. Events include: • Start locking. • Complete locking. • Start lock waiting. • Complete lock waiting. • Start unlocking. • Complete unlocking. • Start lightweight lock waiting. • Complete lightweight lock waiting.
is_slow_sql	boolean	Whether the SQL statement is a slow SQL statement.
trace_id	text	Driver-specific trace ID, which is associated with an application request.
advise	text	Risk information that may cause the SQL statement to be a slow SQL statement. **NOTE** If the statement is executed in SQLBypass mode (you can run the explain command to view the plan. For details, see the description of the enable_opfusion parameter), there is no slow SQL risk and the information in this field may be missing.

Name	Type	Description
parent_unique_sql_id	bigint	Normalized SQL ID of the outer SQL statement. For statements executed in a stored procedure, the value is the normalized SQL ID of the statement that invokes the stored procedure. For statements outside the stored procedure, the value is **0**.
finish_status	text	Statement completion status. The default value is **normal**. **cancelled** indicates cancellation or termination.

13.2.9.8 GS_SLOW_QUERY_INFO

GS_SLOW_QUERY_INFO displays the slow query information that has been dumped on the current node, as described in **Table 13-107**. Data is dumped from the kernel to this system catalog. If the GUC parameter **enable_resource_record** is set to **on**, the system imports the query information from the kernel to GS_WLM_SESSION_QUERY_INFO_ALL every 3 minutes. This operation occupies storage space and affects performance. You can check GS_SLOW_QUERY_INFO to view the slow query information that has been dumped. This view has been discarded in this version.

Table 13-107 GS_SLOW_QUERY_INFO columns

Name	Type	Description
dbname	text	Database name
schemaname	text	Schema name
nodename	text	Node name
username	text	Username
queryid	bigint	Normalization ID
query	text	Query statement
start_time	timestamp with time zone	Execution start time
finish_time	timestamp with time zone	Execution end time
duration	bigint	Execution duration (unit: ms)
query_plan	text	Plan information

Name	Type	Description
n_returned_rows	bigint	Number of rows in the result set returned by the SELECT statement
n_tuples_fetched	bigint	Number of rows randomly scanned
n_tuples_returned	bigint	Number of rows sequentially scanned
n_tuples_inserted	bigint	Number of rows inserted
n_tuples_updated	bigint	Number of rows updated
n_tuples_deleted	bigint	Number of rows deleted
n_blocks_fetched	bigint	Number of cache loading times
n_blocks_hit	bigint	Cache hits
db_time	bigint	Valid DB time, which is accumulated if multiple threads are involved (unit: μs)
cpu_time	bigint	CPU time (unit: μs)
execution_time	bigint	Execution time in the executor (unit: μs)
parse_time	bigint	SQL parsing time (unit: μs)
plan_time	bigint	SQL plan generation time (unit: μs)
rewrite_time	bigint	SQL rewriting time (unit: μs)
pl_execution_time	bigint	Execution time of PL/pgSQL (unit: μs)
pl_compilation_time	bigint	Compilation time of PL/pgSQL (unit: μs)
net_send_time	bigint	Network time (unit: μs)
data_io_time	bigint	I/O time (unit: μs)

13.2.9.9 GS_SLOW_QUERY_HISTORY

GS_SLOW_QUERY_HISTORY displays the slow query information that is not dumped on the current node. For details, see **GS_SLOW_QUERY_INFO**. This is discarded in the current version.

13.2.9.10 GLOBAL_SLOW_QUERY_HISTORY

GS_SLOW_QUERY_HISTORY displays the slow query information that is not dumped on all nodes. This view is discarded in this version. For details, see GS_SLOW_QUERY_INFO.

13.2.9.11 GLOBAL_SLOW_QUERY_INFO

GS_SLOW_QUERY_HISTORY displays the slow query information that has been dumped on all nodes. This view is discarded in this version. For details, see GS_SLOW_QUERY_INFO.

13.2.10 Cache/IO

13.2.10.1 STATIO_USER_TABLES

STATIO_USER_TABLES displays I/O status information about all user relationship tables in the namespace, as described in **Table 13-108**.

Table 13-108 STATIO_USER_TABLES columns

Name	Type	Description
relid	oid	Table OID.
schemaname	name	Name of the schema that contains the table.
relname	name	Table name.
heap_blks_read	bigint	Number of disk blocks read from the table.
heap_blks_hit	bigint	Number of cache hits in the table.
idx_blks_read	bigint	Number of disk blocks read from indexes in the table.
idx_blks_hit	bigint	Number of cache hits in indexes in the table.
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in the table.
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in the table.
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in the table.
tidx_blks_hit	bigint	Number of buffer-hits in the TOAST table index (if any) in the table.

Name	Type	Description
last_updated	timestamp with time zone	Time when the monitoring data of the object in the view is updated for the last time.

13.2.10.2 SUMMARY_STATIO_USER_TABLES

SUMMARY_STATIO_USER_TABLES displays I/O status information about all user relationship tables in namespaces in the database, as described in **Table 13-109**.

Table 13-109 SUMMARY_STATIO_USER_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that contains the table.
relname	name	Table name.
heap_blks_read	numeric	Number of disk blocks read from the table.
heap_blks_hit	numeric	Number of cache hits in the table
idx_blks_read	numeric	Number of disk blocks read from indexes in the table.
idx_blks_hit	numeric	Number of cache hits in indexes in the table.
toast_blks_read	numeric	Number of disk blocks read from the TOAST table (if any) in the table
toast_blks_hit	numeric	Number of buffer hits in the TOAST table (if any) in the table
tidx_blks_read	numeric	Number of disk blocks read from the TOAST table index (if any) in the table
tidx_blks_hit	numeric	Number of buffer-hits in the TOAST table index (if any) in the table

13.2.10.3 GLOBAL_STATIO_USER_TABLES

GLOBAL_STATIO_USER_TABLES displays I/O status information about all user relationship tables in namespaces on each node, as described in **Table 13-110**.

Table 13-110 GLOBAL_STATIO_USER_TABLES columns

Name	Type	Description
node_name	name	Node name

Name	Type	Description
relid	oid	Table OID
schemaname	name	Name of the schema that contains the table
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from the table
heap_blks_hit	bigint	Number of cache hits in the table
idx_blks_read	bigint	Number of disk blocks read from all indexes in the table
idx_blks_hit	bigint	Number of cache hits in the table
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in the table
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in the table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in the table
tidx_blks_hit	bigint	Number of buffer-hits in the TOAST table index (if any) in the table

13.2.10.4 STATIO_USER_INDEXES

STATIO_USER_INDEXES displays I/O status information about all user relationship table indexes in namespaces on the current node, as described in **Table 13-111**.

Table 13-111 STATIO_USER_INDEXES columns

Name	Type	Description
relid	oid	OID of the table that the index is created for
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	bigint	Number of disk blocks read from the index

Name	Type	Description
idx_blks_hit	bigint	Number of cache hits in the index
last_updated	timestamp with time zone	Time when the monitoring data of the object in the view is updated for the last time.

13.2.10.5 SUMMARY_STATIO_USER_INDEXES

SUMMARY_STATIO_USER_INDEXES displays I/O status information about all user relationship table indexes in namespaces in the database, as described in **Table 13-112**.

Table 13-112 SUMMARY_STATIO_USER_INDEXES columns

Name	Type	Description
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	numeric	Number of disk blocks read from the index
idx_blks_hit	numeric	Number of cache hits in the index

13.2.10.6 GLOBAL_STATIO_USER_INDEXES

GLOBAL_STATIO_USER_INDEXES displays I/O status information about all user relationship table indexes in namespaces on each node, as described in **Table 13-113**.

Table 13-113 GLOBAL_STATIO_USER_INDEXES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the table that the index is created for
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name

Name	Type	Description
idx_blks_read	numeric	Number of disk blocks read from the index
idx_blks_hit	numeric	Number of cache hits in the index

13.2.10.7 STATIO_USER_SEQUENCES

STATIO_USER_SEQUENCE displays I/O status information about all user relationship table sequences in namespaces on the current node, as described in **Table 13-114**.

Table 13-114 STATIO_USER_SEQUENCE columns

Name	Type	Description
relid	oid	OID of the sequence
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Number of cache hits in the sequence

13.2.10.8 SUMMARY_STATIO_USER_SEQUENCES

SUMMARY_STATIO_USER_SEQUENCES displays I/O status information about all user relationship table sequences in namespaces in the database, as described in **Table 13-115**.

Table 13-115 SUMMARY_STATIO_USER_SEQUENCES columns

Name	Type	Description
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	numeric	Number of disk blocks read from the sequence
blks_hit	numeric	Number of cache hits in the sequence

13.2.10.9 GLOBAL_STATIO_USER_SEQUENCES

GLOBAL_STATIO_USER_SEQUENCES displays I/O status information about all user relationship table sequences in namespaces on each node, as described in **Table 13-116**.

Table 13-116 GLOBAL_STATIO_USER_SEQUENCES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the sequence
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Number of cache hits in the sequence

13.2.10.10 STATIO_SYS_TABLES

STATIO_SYS_TABLES displays I/O status information about all system catalogs in the current namespace, as described in **Table 13-117**.

Table 13-117 STATIO_SYS_TABLES columns

Name	Type	Description
relid	oid	Table OID.
schemaname	name	Name of the schema that contains the table.
relname	name	Table name.
heap_blks_read	bigint	Number of disk blocks read from the table.
heap_blks_hit	bigint	Number of cache hits in the table.
idx_blks_read	bigint	Number of disk blocks read from indexes in the table.
idx_blks_hit	bigint	Number of cache hits in indexes in the table.
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in the table.
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in the table.

Name	Type	Description
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in the table.
tidx_blks_hit	bigint	Number of buffer-hits in the TOAST table index (if any) in the table.
last_updated	timestamp with time zone	Time when the monitoring data of the object in the view is updated for the last time.

13.2.10.11 SUMMARY_STATIO_SYS_TABLES

SUMMARY_STATIO_SYS_TABLES displays I/O status information about all system catalogs in namespaces in the database, as described in **Table 13-118**.

Table 13-118 SUMMARY_STATIO_SYS_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that contains the table.
relname	name	Table name.
heap_blks_read	numeric	Number of disk blocks read from the table.
heap_blks_hit	numeric	Number of cache hits in the table
idx_blks_read	numeric	Number of disk blocks read from indexes in the table.
idx_blks_hit	numeric	Number of cache hits in indexes in the table.
toast_blks_read	numeric	Number of disk blocks read from the TOAST table (if any) in the table
toast_blks_hit	numeric	Number of buffer hits in the TOAST table (if any) in the table
tidx_blks_read	numeric	Number of disk blocks read from the TOAST table index (if any) in the table
tidx_blks_hit	numeric	Number of buffer-hits in the TOAST table index (if any) in the table

13.2.10.12 GLOBAL_STATIO_SYS_TABLES

GLOBAL_STATIO_SYS_TABLES displays I/O status information about all system catalogs in namespaces on each node, as described in **Table 13-119**.

Table 13-119 GLOBAL_STATIO_SYS_TABLES columns

Name	Type	Description
node_name	name	Node name
relid	oid	Table OID
schemaname	name	Name of the schema that contains the table
relname	name	Table name
heap_blks_read	bigint	Number of disk blocks read from the table
heap_blks_hit	bigint	Number of cache hits in the table
idx_blks_read	bigint	Number of disk blocks read from all indexes in the table
idx_blks_hit	bigint	Number of cache hits in the table
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in the table
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in the table
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in the table
tidx_blks_hit	bigint	Number of buffer-hits in the TOAST table index (if any) in the table

13.2.10.13 STATIO_SYS_INDEXES

STATIO_SYS_INDEXES displays the I/O status information about all system catalog indexes in the current namespace, as described in **Table 13-120**.

Table 13-120 STATIO_SYS_INDEXES columns

Name	Type	Description
relid	oid	OID of the table that the index is created for
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name

Name	Type	Description
idx_blks_read	bigint	Number of disk blocks read from the index
idx_blks_hit	bigint	Number of cache hits in the index
last_updated	timestamp with time zone	Time when the monitoring data of the object in the view is updated for the last time.

13.2.10.14 SUMMARY_STATIO_SYS_INDEXES

SUMMARY_STATIO_SYS_INDEXES displays I/O status information about all system catalog indexes in namespaces in the database, as described in **Table 13-121**.

Table 13-121 SUMMARY_STATIO_SYS_INDEXES columns

Name	Type	Description
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	numeric	Number of disk blocks read from the index
idx_blks_hit	numeric	Number of cache hits in the index

13.2.10.15 GLOBAL_STATIO_SYS_INDEXES

GLOBAL_STATIO_SYS_INDEXES displays I/O status information about all system catalog indexes in namespaces on each node, as described in **Table 13-122**.

Table 13-122 GLOBAL_STATIO_SYS_INDEXES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the table that the index is created for
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for

Name	Type	Description
indexrelname	name	Index name
idx_blks_read	numeric	Number of disk blocks read from the index
idx_blks_hit	numeric	Number of cache hits in the index

13.2.10.16 STATIO_SYS_SEQUENCES

STATIO_SYS_SEQUENCES shows the I/O status information about all the system sequences in the current namespace, as described in **Table 13-123**.

Table 13-123 STATIO_SYS_SEQUENCES columns

Name	Type	Description
relid	oid	OID of the sequence
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Number of cache hits in the sequence

13.2.10.17 SUMMARY_STATIO_SYS_SEQUENCES

SUMMARY_STATIO_SYS_SEQUENCES displays I/O status information about all system sequences in namespaces in the database, as described in **Table 13-124**.

Table 13-124 SUMMARY_STATIO_SYS_SEQUENCES columns

Name	Type	Description
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	numeric	Number of disk blocks read from the sequence
blks_hit	numeric	Number of cache hits in the sequence

13.2.10.18 GLOBAL_STATIO_SYS_SEQUENCES

GLOBAL_STATIO_SYS_SEQUENCES displays I/O status information about all system sequences in namespaces on each node, as described in **Table 13-125**.

Table 13-125 GLOBAL_STATIO_SYS_SEQUENCES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the sequence
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Number of cache hits in the sequence

13.2.10.19 STATIO_ALL_TABLES

STATIO_ALL_TABLES contains one row for each table (including TOAST tables) in the current database, showing I/O statistics about specific tables, as described in **Table 13-126**.

Table 13-126 STATIO_ALL_TABLES columns

Name	Type	Description
relid	oid	Table OID.
schemaname	name	Name of the schema that contains the table.
relname	name	Table name.
heap_blks_read	bigint	Number of disk blocks read from the table.
heap_blks_hit	bigint	Number of cache hits in the table.
idx_blks_read	bigint	Number of disk blocks read from indexes in the table.
idx_blks_hit	bigint	Number of cache hits in indexes in the table.
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in the table.
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in the table.

Name	Type	Description
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in the table.
tidx_blks_hit	bigint	Number of buffer-hits in the TOAST table index (if any) in the table.
last_updated	timestamp with time zone	Time when the monitoring data of the object in the view is updated for the last time.

13.2.10.20 SUMMARY_STATIO_ALL_TABLES

SUMMARY_STATIO_ALL_TABLES contains I/O statistics about each table (including TOAST tables) in databases in the database, as described in **Table 13-127**.

Table 13-127 SUMMARY_STATIO_ALL_TABLES columns

Name	Type	Description
schemaname	name	Name of the schema that contains the table.
relname	name	Table name.
heap_blks_read	numeric	Number of disk blocks read from the table.
heap_blks_hit	numeric	Number of cache hits in the table.
idx_blks_read	numeric	Number of disk blocks read from all indexes in the table.
idx_blks_hit	numeric	Number of cache hits in indexes in the table.
toast_blks_read	numeric	Number of disk blocks read from the TOAST table (if any) in the table.
toast_blks_hit	numeric	Number of buffer hits in the TOAST table (if any) in the table.
tidx_blks_read	numeric	Number of disk blocks read from the TOAST table index (if any) in the table.
tidx_blks_hit	numeric	Number of buffer-hits in the TOAST table index (if any) in the table.
last_updated	timestamp with time zone	Time when the monitoring data of the table in the view is updated for the last time.

13.2.10.21 GLOBAL_STATIO_ALL_TABLES

GLOBAL_STATIO_ALL_TABLES contains I/O statistics about each table (including TOAST tables) in databases on each node, as described in **Table 13-128**.

Table 13-128 GLOBAL_STATIO_ALL_TABLES columns

Name	Type	Description
node_name	name	Node name.
relid	oid	Table OID.
schemaname	name	Name of the schema that contains the table.
relname	name	Table name.
heap_blks_read	bigint	Number of disk blocks read from the table.
heap_blks_hit	bigint	Number of cache hits in the table.
idx_blks_read	bigint	Number of disk blocks read from all indexes in the table.
idx_blks_hit	bigint	Number of cache hits in indexes in the table.
toast_blks_read	bigint	Number of disk blocks read from the TOAST table (if any) in the table.
toast_blks_hit	bigint	Number of buffer hits in the TOAST table (if any) in the table.
tidx_blks_read	bigint	Number of disk blocks read from the TOAST table index (if any) in the table.
tidx_blks_hit	bigint	Number of buffer-hits in the TOAST table index (if any) in the table.
last_updated	timestamp with time zone	Time when the monitoring data of the table in the view is updated for the last time.

13.2.10.22 STATIO_ALL_INDEXES

STATIO_ALL_INDEXES contains one row for each index in the current database, showing I/O statistics about specific indexes, as described in **Table 13-129**.

Table 13-129 STATIO_ALL_INDEXES columns

Name	Type	Description
relid	oid	OID of the table that the index is created for

Name	Type	Description
indexrelid	oid	OID of the index
schemaname	name	Name of the schema that the index is in
relname	name	Name of the table that the index is created for
indexrelname	name	Index name
idx_blks_read	bigint	Number of disk blocks read from the index
idx_blks_hit	bigint	Number of cache hits in the index
last_updated	timestamp with time zone	Time when the monitoring data of the object in the view is updated for the last time.

13.2.10.23 SUMMARY_STATIO_ALL_INDEXES

SUMMARY_STATIO_ALL_INDEXES contains every row of each index in a database, showing I/O statistics about specific indexes, as described in **Table 13-130**.

Table 13-130 SUMMARY_STATIO_ALL_INDEXES columns

Name	Type	Description
schemaname	name	Name of the schema that the index is in.
relname	name	Name of the table that the index is created for.
indexrelname	name	Index name.
idx_blks_read	numeric	Number of disk blocks read from the index.
idx_blks_hit	numeric	Number of cache hits in the index.
last_updated	timestamp with time zone	Time when the monitoring data of the index in the view is updated for the last time.

13.2.10.24 GLOBAL_STATIO_ALL_INDEXES

GLOBAL_STATIO_ALL_INDEXES contains one row for each index in databases on each node, showing I/O statistics about specific indexes, as described in **Table 13-131**.

Table 13-131 GLOBAL_STATIO_ALL_INDEXES columns

Name	Type	Description
node_name	name	Node name.
relid	oid	OID of the table for this index.
indexrelid	oid	OID of the index.
schemaname	name	Name of the schema that the index is in.
relname	name	Name of the table that the index is created for.
indexrelname	name	Index name.
idx_blks_read	numeric	Number of disk blocks read from the index.
idx_blks_hit	numeric	Number of cache hits in the index.
last_updated	timestamp with time zone	Time when the monitoring data of the index in the view is updated for the last time.

13.2.10.25 STATIO_ALL_SEQUENCES

STATIO_ALL_SEQUENCES contains one row for each sequence in the current database, showing I/O statistics about specific sequences, as described in **Table 13-132**.

Table 13-132 STATIO_ALL_SEQUENCES columns

Name	Type	Description
relid	oid	OID of this sequence
schemaname	name	Name of the schema where the sequence is in
relname	name	Sequence name
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Cache hits in the sequence

13.2.10.26 SUMMARY_STATIO_ALL_SEQUENCES

SUMMARY_STATIO_ALL_SEQUENCES contains each row for all sequences in the entire database, showing I/O statistics about specific sequences, as described in **Table 13-133**.

Table 13-133 SUMMARY_STATIO_ALL_SEQUENCES columns

Name	Type	Description
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	numeric	Number of disk blocks read from the sequence
blks_hit	numeric	Number of cache hits in the sequence

13.2.10.27 GLOBAL_STATIO_ALL_SEQUENCES

GLOBAL_STATIO_ALL_SEQUENCES contains every row of each sequence in databases on each node, showing I/O statistics about specific sequences, as described in **Table 13-134**.

Table 13-134 GLOBAL_STATIO_ALL_SEQUENCES columns

Name	Type	Description
node_name	name	Node name
relid	oid	OID of the sequence
schemaname	name	Name of the schema that the sequence is in
relname	name	Sequence name
blks_read	bigint	Number of disk blocks read from the sequence
blks_hit	bigint	Number of cache hits in the sequence

13.2.11 Utility

13.2.11.1 REPLICATION_STAT

REPLICATION_STAT describes information about log synchronization status, such as the locations where the sender sends logs and where the receiver receives logs, as described in **Table 13-135**.

Table 13-135 REPLICATION_STAT columns

Name	Type	Description
pid	bigint	Process ID of the thread
usesysid	oid	User system ID

Name	Type	Description
usename	name	Username
application_name	text	Program name
client_addr	inet	Client address
client_hostname	text	Client name
client_port	integer	Port of the client
backend_start	timestamp with time zone	Start time of the program
state	text	Log replication state: • Catch-up state • Consistent streaming state
sender_sent_location	text	Location where the sender sends logs
receiver_write_location	text	Location where the receiver writes logs
receiver_flush_location	text	Location where the receiver flushes logs
receiver_replay_location	text	Location where the receiver replays logs
sync_priority	integer	Priority of synchronous duplication (**0** indicates asynchronization.)
sync_state	text	Synchronization state: • Asynchronous replication • Synchronous replication • Potential synchronization

13.2.11.2 GLOBAL_REPLICATION_STAT

GLOBAL_REPLICATION_STAT displays information about log synchronization status on each node, such as the locations where the sender sends logs and where the receiver receives logs, as described in **Table 13-136**.

Table 13-136 GLOBAL_REPLICATION_STAT columns

Name	Type	Description
node_name	name	Node name.
pid	bigint	PID of the thread
usesysid	oid	User system ID

Name	Type	Description
usename	name	Username
application_name	text	Program name
client_addr	inet	Client address
client_hostname	text	Client name
client_port	integer	Port of the client
backend_start	timestamp with time zone	Start time of the program
state	text	Log replication state: • Catch-up state • Consistent streaming state
sender_sent_location	text	Location where the transmit sends logs
receiver_write_location	text	Location where the receive end writes logs
receiver_flush_location	text	Location where the receive end flushes logs
receiver_replay_location	text	Location where the receive end replays logs
sync_priority	integer	Priority of synchronous duplication (**0** indicates asynchronization.)
sync_state	text	Synchronization state: • Asynchronous replication • Synchronous replication • Potential synchronization

13.2.11.3 REPLICATION_SLOTS

REPLICATION_SLOTS displays replication slot information, as described in **Table 13-137**.

Table 13-137 REPLICATION_SLOTS columns

Name	Type	Description
slot_name	text	Replication slot name.
plugin	text	Name of the output plug-in corresponding to the logical replication slot.

Name	Type	Description
slot_type	text	Replication slot type. • **physical**: physical replication slot. • **logical**: logical replication slot.
datoid	oid	OID of the database where the replication slot resides.
database	name	Name of the database where the replication slot resides.
active	boolean	Specifies whether the replication slot is activated. • **t** (true): yes • **f** (false): no
xmin	xid	XID of the earliest transaction that the database must reserve for the replication slot.
catalog_xmin	xid	XID of the earliest system catalog-involved transaction that the database must reserve for the logical replication slot.
restart_lsn	text	Physical location of the earliest Xlog required by the replication slot.
dummy_standby	boolean	Reserved parameter.

13.2.11.4 GLOBAL_REPLICATION_SLOTS

GLOBAL_REPLICATION_SLOTS displays replication slot information of each node in the database, as described in **Table 13-138**.

Table 13-138 GLOBAL_REPLICATION_SLOTS columns

Name	Type	Description
node_name	name	Node name.
slot_name	text	Replication slot name.
plugin	text	Name of the output plug-in corresponding to the logical replication slot.
slot_type	text	Replication slot type. • **physical**: physical replication slot. • **logical**: logical replication slot.
datoid	oid	OID of the database where the replication slot resides.

Name	Type	Description
database	name	Name of the database where the replication slot resides.
active	boolean	Determines whether the replication slot is activated. • **t** (true): yes • **f** (false): no
x_min	xid	XID of the earliest transaction that the database must reserve for the replication slot.
catalog_xmin	xid	XID of the earliest system catalog-involved transaction that the database must reserve for the logical replication slot.
restart_lsn	text	Physical location of the earliest Xlog required by the replication slot.
dummy_standby	boolean	Reserved parameter.

13.2.11.5 BGWRITER_STAT

BGWRITER_STAT displays statistics about the background writer thread's activities, as described in **Table 13-139**.

Table 13-139 BGWRITER_STAT columns

Name	Type	Description
checkpoints_timed	bigint	Number of scheduled checkpoints that have been performed.
checkpoints_req	bigint	Number of requested checkpoints that have been performed.
checkpoint_write_time	double precision	Total time that has been spent in the portion of checkpoint processing where files are written to disk (unit: ms).
checkpoint_sync_time	double precision	Total time that has been spent in the portion of checkpoint processing where files are synchronized to disk (unit: ms).
buffers_checkpoint	bigint	Number of buffers written during checkpoints.
buffers_clean	bigint	Number of buffers written by the background writer thread.

Name	Type	Description
maxwritten_clean	bigint	Number of times the background writer thread stopped a cleaning scan because it had written too many buffers.
buffers_backend	bigint	Number of buffers written directly by a backend.
buffers_backend_fsync	bigint	Number of times a backend had to execute its own fsync call (normally the background writer thread handles those even when the backend does its own write).
buffers_alloc	bigint	Number of buffers allocated.
stats_reset	timestamp with time zone	Time at which these statistics were last reset.

13.2.11.6 GLOBAL_BGWRITER_STAT

GLOBAL_BGWRITER_STAT displays statistics about the background writer thread's activities on each node, as described in **Table 13-140**.

Table 13-140 GLOBAL_BGWRITER_STAT columns

Name	Type	Description
node_name	name	Node name.
checkpoints_timed	bigint	Number of scheduled checkpoints that have been performed.
checkpoints_req	bigint	Number of requested checkpoints that have been performed.
checkpoint_write_time	double precision	Total time that has been spent in the portion of checkpoint processing where files are written to disk (unit: ms).
checkpoint_sync_time	double precision	Total time that has been spent in the portion of checkpoint processing where files are synchronized to disk (unit: ms).
buffers_checkpoint	bigint	Number of buffers written during checkpoints.
buffers_clean	bigint	Number of buffers written by the background writer thread.

Name	Type	Description
maxwritten_clean	bigint	Number of times the background writer thread stopped a cleaning scan because it had written too many buffers.
buffers_backend	bigint	Number of buffers written directly by a backend.
buffers_backend_fsync	bigint	Number of times a backend had to execute its own fsync call (normally the background writer thread handles those even when the backend does its own write).
buffers_alloc	bigint	Number of buffers allocated.
stats_reset	timestamp with time zone	Time at which these statistics were last reset.

13.2.11.7 GLOBAL_CKPT_STATUS

GLOBAL_CKPT_STATUS displays the checkpoint information and log flushing information about all instances in the database, as described in **Table 13-141**.

Table 13-141 GLOBAL_CKPT_STATUS columns

Name	Type	Description
node_name	text	Node name.
ckpt_redo_point	text	Checkpoint of the current instance
ckpt_clog_flush_num	bigint	Number of Clog flushing pages from the startup time to the current time
ckpt_csnlog_flush_num	bigint	Number of CSN log flushing pages from the startup time to the current time
ckpt_multixact_flush_num	bigint	Number of MultiXact flushing pages from the startup time to the current time
ckpt_predicate_flush_num	bigint	Number of predicate flushing pages from the startup time to the current time
ckpt_twophase_flush_num	bigint	Number of two-phase flushing pages from the startup time to the current time

13.2.11.8 GLOBAL_DOUBLE_WRITE_STATUS

GLOBAL_DOUBLE_WRITE_STATUS displays doublewrite file status of all instances in the database, as described in **Table 13-142**. It consists of the local_double_write_stat view of each node. Columns in the view on each node are the same.

Table 13-142 GLOBAL_DOUBLE_WRITE_STATUS columns

Name	Type	Description
node_name	text	Node name.
curr_dwn	bigint	Sequence number of the doublewrite file
curr_start_page	bigint	Start page for restoring the doublewrite file
file_trunc_num	bigint	Number of times that the doublewrite file is reused
file_reset_num	bigint	Number of reset times after the doublewrite file is full
total_writes	bigint	Total number of I/Os of the doublewrite file
low_threshold_writes	bigint	Number of I/Os for writing doublewrite files with low efficiency (the number of I/O flushing pages at a time is less than 16)
high_threshold_writes	bigint	Number of I/Os for writing doublewrite files with high efficiency (the number of I/O flushing pages at a time is more than 421)
total_pages	bigint	Total number of pages that are flushed to the doublewrite file area
low_threshold_pages	bigint	Number of pages that are flushed with low efficiency
high_threshold_pages	bigint	Number of pages that are flushed with high efficiency
file_id	bigint	ID of the current doublewrite file

13.2.11.9 GLOBAL_PAGEWRITER_STATUS

GLOBAL_PAGEWRITER_STATUS displays the page flushing information and checkpoint information about all instances in the database, as described in **Table 13-143**.

Table 13-143 GLOBAL_PAGEWRITER_STATUS columns

Name	Type	Description
node_name	text	Node name.

Name	Type	Description
pgwr_actual_flush_total_num	bigint	Total number of dirty pages flushed from the startup to the current time.
pgwr_last_flush_num	integer	Number of dirty pages flushed in the previous batch
remain_dirty_page_num	bigint	Estimated number of remaining dirty pages.
queue_head_page_rec_lsn	text	**recovery_lsn** of the first dirty page in the dirty page queue of the current instance
queue_rec_lsn	text	**recovery_lsn** of the dirty page queue of the current instance
current_xlog_insert_lsn	text	The write position of Xlogs in the current instance
ckpt_redo_point	text	Checkpoint of the current instance

13.2.11.10 GLOBAL_RECORD_RESET_TIME

GLOBAL_RECORD_RESET_TIME displays statistics about reset time in the database. Restart, primary/standby switchover, and database deletion will cause the time to be reset, as described in **Table 13-144**.

Table 13-144 GLOBAL_RECORD_RESET_TIME columns

Name	Type	Description
node_name	text	Node name.
reset_time	timestamp with time zone	Time to be reset

13.2.11.11 GLOBAL_REDO_STATUS

GLOBAL_REDO_STATUS displays the log replay status of all instances in the database, as described in **Table 13-145**.

Table 13-145 GLOBAL_REDO_STATUS columns

Name	Type	Description
node_name	text	Node name.
redo_start_ptr	bigint	Start point for replaying the instance logs
redo_start_time	bigint	Start time (UTC) when the instance logs are replayed

Name	Type	Description
redo_done_time	bigint	End time (UTC) when the instance logs are replayed
curr_time	bigint	Current time (UTC) of the instance
min_recovery_point	bigint	Position of the minimum consistency point for the instance logs
read_ptr	bigint	Position for reading the instance logs
last_replayed_read_ptr	bigint	Position for replaying the instance logs
recovery_done_ptr	bigint	Replay position after the instance is started
read_xlog_io_counter	bigint	Number of I/Os when the current instance reads and replays logs
read_xlog_io_total_dur	bigint	Total I/O latency when the current instance reads and replays logs
read_data_io_counter	bigint	Number of data page I/O reads during replay in the current instance
read_data_io_total_dur	bigint	Total I/O latency of data page reads during replay in the current instance
write_data_io_counter	bigint	Number of data page I/O writes during replay in the current instance
write_data_io_total_dur	bigint	Total I/O latency of data page writes during replay in the current instance
process_pending_counter	bigint	Number of synchronization times of log distribution threads during replay in the instance
process_pending_total_dur	bigint	Total synchronization latency of log distribution threads during replay in the instance
apply_counter	bigint	Number of synchronization times of replay threads during replay in the instance
apply_total_dur	bigint	Total synchronization latency of replay threads during replay in the instance
speed	bigint	Log replay rate of the current instance
local_max_ptr	bigint	Maximum number of replay logs received by the local host after the instance is started
primary_flush_ptr	bigint	Log point where the host flushes logs to a disk

Name	Type	Description
worker_info	text	Replay thread information of the instance. If concurrent replay is not enabled, the value is **NULL**.

13.2.11.12 GLOBAL_RECOVERY_STATUS

GLOBAL_RECOVERY_STATUS displays log flow control information about the primary and standby nodes, as described in **Table 13-146**.

Table 13-146 GLOBAL_RECOVERY_STATUS columns

Name	Type	Description
node_name	text	Node name (including the primary and standby nodes)
standby_node_name	text	Name of the standby node
source_ip	text	IP address of the primary node
source_port	integer	Port number of the primary node
dest_ip	text	IP address of the standby node
dest_port	integer	Port number of the standby node
current_rto	bigint	Current log flow control time of the standby node (unit: s)
target_rto	bigint	Expected flow control time of the standby node specified by the corresponding GUC parameter (unit: s)
current_sleep_time	bigint	Sleep time required to achieve the expected flow control time (unit: μs)

13.2.11.13 CLASS_VITAL_INFO

CLASS_VITAL_INFO is used to check whether the OIDs of the same table or index are consistent for WDR snapshots, as described in **Table 13-147**.

Table 13-147 CLASS_VITAL_INFO columns

Name	Type	Description
relid	oid	Table OID.
schemaname	name	Schema name.

Name	Type	Description
relname	name	Table name.
relkind	"char"	Object type. Its value can be: • **r**: ordinary table • **t**: TOAST table • **i**: index
last_updated	timestamp with time zone	Time when the monitoring data of the object in the view is updated for the last time.

13.2.11.14 USER_LOGIN

USER_LOGIN records the number of user logins and logouts, as described in **Table 13-148**.

Table 13-148 USER_LOGIN columns

Name	Type	Description
node_name	text	Database process name
user_name	text	Username
user_id	integer	User OID (Its value is the same as that of **oid** in **pg_authid**.)
login_counter	bigint	Number of logins
logout_counter	bigint	Number of logouts

13.2.11.15 SUMMARY_USER_LOGIN

SUMMARY_USER_LOGIN records information about user logins and logouts on the primary database node, as described in **Table 13-149**.

Table 13-149 SUMMARY_USER_LOGIN columns

Name	Type	Description
node_name	text	Database process name
user_name	text	Username

Name	Type	Description
user_id	integer	User OID (Its value is the same as that of **oid** in **pg_authid**.)
login_counter	bigint	Number of logins
logout_counter	bigint	Number of logouts

13.2.11.16 GLOBAL_SINGLE_FLUSH_DW_STATUS

GLOBAL_SINGLE_FLUSH_DW_STATUS displays information about doublewrite files eliminated on a single page of all instances in the database, as described in **Table 13-150**. In the displayed information, the information before the slash (/) indicates the page flushing status of the first version, and the information after the slash (/) indicates the page flushing status of the second version.

Table 13-150 GLOBAL_SINGLE_FLUSH_DW_STATUS columns

Name	Type	Description
node_name	text	Instance name
curr_dwn	text	Sequence number of the doublewrite file
curr_start_page	text	Start position of the current doublewrite file
total_writes	text	Total number of data write pages in the current doublewrite file
file_trunc_num	text	Number of times that the doublewrite file is reused
file_reset_num	text	Number of reset times after the doublewrite file is full

13.2.11.17 GLOBAL_CANDIDATE_STATUS

GLOBAL_CANDIDATE_STATUS displays the number of candidate buffers and buffer eviction information of all instances in the database, as described in **Table 13-151**.

Table 13-151 GLOBAL_GET_BGWRITER_STATUS columns

Name	Type	Description
node_name	text	Node name
candidate_slots	integer	Number of pages in the candidate buffer chain of the current normal buffer pool

Name	Type	Description
get_buf_from_list	bigint	Number of times that pages are obtained from the candidate buffer chain during buffer eviction in the current normal buffer pool
get_buf_clock_sweep	bigint	Number of times that pages are obtained from the original eviction solution during buffer eviction in the current normal buffer pool
seg_candidate_slots	integer	Number of pages in the candidate buffer chain of the current segment buffer pool
seg_get_buf_from_list	bigint	Number of times that pages are obtained from the candidate buffer chain during buffer eviction in the current segment buffer pool
seg_get_buf_clock_sweep	bigint	Number of times that pages are obtained from the original eviction solution during buffer eviction in the current segment buffer pool

13.2.11.18 PARALLEL_DECODE_STATUS

PARALLEL_DECODE_STATUS displays parallel decoding information about replication slots on the current node, as described in **Table 13-152**.

Table 13-152 PARALLEL_DECODE_STATUS columns

Name	Type	Description
slot_name	text	Specifies the replication slot name.
parallel_decode_num	integer	Specifies the number of parallel decoder threads of the replication slot.
read_change_queue_length	text	Concatenates the current length of the log queue read by each decoder thread and then outputs the result.
decode_change_queue_length	text	Concatenates the current length of the decoding result queue of each decoder thread and then outputs the result.
reader_lsn	text	Specifies the location of the log read by the current reader thread.

Name	Type	Description
working_txn_cnt	bigint	Specifies the number of transactions that are being concatenated in the current sender thread.
working_txn_memory	bigint	Specifies the total memory occupied by concatenation transactions in the sender thread, in bytes.
decoded_time	timestamptz	Specifies the time of the latest WAL decoded by the replication slot.

13.2.11.19 GLOBAL_PARALLEL_DECODE_STATUS

GLOBAL_PARALLEL_DECODE_STATUS displays parallel decoding information about replication slots on the current node, as described in **Table 13-153**.

Table 13-153 GLOBAL_PARALLEL_DECODE_STATUS columns

Name	Type	Description
node_name	name	Specifies the node name.
slot_name	text	Specifies the replication slot name.
parallel_decode_num	integer	Specifies the number of parallel decoder threads of the replication slot.
read_change_queue_length	text	Concatenates the current length of the log queue read by each decoder thread and then outputs the result.
decode_change_queue_length	text	Concatenates the current length of the decoding result queue of each decoder thread and then outputs the result.
reader_lsn	text	Specifies the location of the log read by the current reader thread.
working_txn_cnt	bigint	Specifies the number of transactions that are being concatenated in the current sender thread.
working_txn_memory	bigint	Total memory occupied by concatenation transactions in the sender thread, in bytes.
decoded_time	timestamptz	Specifies the time of the latest WAL decoded by the replication slot.

13.2.11.20 PARALLEL_DECODE_THREAD_INFO

PARALLEL_DECODE_THREAD_INFO displays information about threads that perform parallel decoding on the current node, as described in **Table 13-154**.

Table 13-154 PARALLEL_DECODE_THREAD_INFO columns

Name	Type	Description
thread_id	bigint	Thread ID.
slot_name	text	Replication slot name.
thread_type	text	Thread type (sender, reader, or decoder).
seq_number	integer	Sequence number (starting from 1) of threads of the same type in the current replication slot.

13.2.11.21 GLOBAL_PARALLEL_DECODE_THREAD_INFO

GLOBAL_PARALLEL_DECODE_THREAD_INFO displays information about parallel decoder threads of replication slots on the current node, as described in **Table 13-155**.

Table 13-155 GLOBAL_PARALLEL_DECODE_THREAD_INFO columns

Name	Type	Description
node_name	name	Node name.
thread_id	bigint	Thread ID.
slot_name	text	Replication slot name.
thread_type	text	Thread type (sender, reader, or decoder).
seq_number	integer	Sequence number (starting from 1) of threads of the same type in the current replication slot.

13.2.11.22 GLOBAL_ADIO_COMPLETER_STATUS

GLOBAL_ADIO_COMPLETER_STATUS displays statistics about the AIO Completer thread in all database instances, as described in **Table 13-156**.

Table 13-156 GLOBAL_ADIO_COMPLETER_STATUS columns

Name	Type	Description
node_name	text	Instance name.
tid	int8	ID of the AIO Completer thread.
thread_type	text	AIO Completer thread type (read or write).
aio_submitted_num	int8	Number of committed asynchronous I/O requests of the AIO Completer thread.
aio_completed_num	int8	Number of completed asynchronous I/O requests of the AIO Completer thread.
aio_incompleted_num	int8	Number of asynchronous I/O requests that are not completed of the AIO Completer thread.
slot_count_left	int8	Indicates the number of idle slots.

13.2.11.23 GLOBAL_AIO_SLOT_USAGE_STATUS

GLOBAL_AIO_SLOT_USAGE_STATUS displays statistics about asynchronous I/O commit slots in all database instances, as described in **Table 13-157**.

Table 13-157 GLOBAL_AIO_SLOT_USAGE_STATUS columns

Name	Type	Description
node_name	text	Instance name.
slot_id	int4	Slot ID.
slot_type	char	Slot type. The value can be **r** (read) or **w** (write).
status	bool	Slot usage status.
buffer_id	int8	Buffer ID corresponding to the slot.
relfilenode_blocknum	text	Position of the physical page where the buffer corresponding to the slot is located.
lsn	int8	LSN corresponding to the page.
submitted_time	int8	Time when a page is committed asynchronously.
elapsed_time	int8	Waiting time of the page.

13.2.12 Lock

13.2.12.1 LOCKS

LOCKS displays information about locks held by each open transaction, as described in **Table 13-158**.

Table 13-158 LOCKS columns

Name	Type	Description
locktype	text	Type of the locked object: **relation**, **extend**, **page**, **tuple**, **transactionid**, **virtualxid**, **object**, **userlock**, or **advisory**
database	oid	OID of the database in which the locked object exists. • The OID is **0** if the object is a shared object. • The OID is **NULL** if the object is a transaction ID.
relation	oid	OID of the relationship targeted by the lock. The value is **NULL** if the object is not a relationship or part of a relationship.
page	integer	Page number targeted by the lock within the relationship. The value is **NULL** if the object is not a relationship page or row page.
tuple	smallint	Row number targeted by the lock within the page. The value is **NULL** if the object is not a row.
bucket	integer	Hash bucket ID.
virtualxid	text	Virtual ID of the transaction targeted by the lock. The value is **NULL** if the object is not a virtual transaction ID.
transactionid	xid	ID of the transaction targeted by the lock. The value is **NULL** if the object is not a transaction ID.
classid	oid	OID of the system catalog that contains the object. The value is **NULL** if the object is not a general database object.
objid	oid	OID of the locked object within its system catalog. The value is **NULL** if the object is not a general database object.
objsubid	smallint	Column number for a column in the table (**0** if the object is of other object type and **NULL** if the object is not a general database object)
virtualtransaction	text	Virtual ID of the transaction holding or awaiting this lock

Name	Type	Description
pid	bigint	Logical ID of the server thread holding or awaiting this lock. The value is **NULL** if the lock is held by a prepared transaction.
sessionid	bigint	ID of the session holding or awaiting this lock The value is **NULL** if the lock is held by a prepared transaction.
mode	text	Lock mode held or desired by this thread
granted	Boolean	• The value is **TRUE** if the lock is a held lock. • The value is **FALSE** if the lock is an awaited lock.
fastpath	Boolean	The value is **TRUE** if the lock is obtained through **fast-path**, and is **FALSE** if the lock is obtained through the main lock table.
locktag	text	Lock information that the session waits for. It can be parsed using the **locktag_decode()** function.
global_sessionid	text	Global session ID

13.2.12.2 GLOBAL_LOCKS

GLOBAL_LOCKS displays information about locks held by open transactions on each node, as described in **Table 13-159**.

Table 13-159 GLOBAL_LOCKS columns

Name	Type	Description
node_name	name	Node name.
locktype	text	Type of the locked object: **relation**, **extend**, **page**, **tuple**, **transactionid**, **virtualxid**, **object**, **userlock**, or **advisory**
database	oid	OID of the database in which the locked object exists. • The OID is **0** if the object is a shared object. • The OID is **NULL** if the object is a transaction ID.
relation	oid	OID of the relationship targeted by the lock. The value is **NULL** if the object is not a relationship or part of a relationship.

Name	Type	Description
page	integer	Page number targeted by the lock within the relationship. The value is **NULL** if the object is not a relationship page or row page.
tuple	smallint	Row number targeted by the lock within the page. The value is **NULL** if the object is not a row.
virtualxid	text	Virtual ID of the transaction targeted by the lock. The value is **NULL** if the object is not a virtual transaction ID.
transactionid	xid	ID of the transaction targeted by the lock. The value is **NULL** if the object is not a transaction ID.
classid	oid	OID of the system catalog that contains the object. The value is **NULL** if the object is not a general database object.
objid	oid	OID of the locked object within its system catalog. The value is **NULL** if the object is not a general database object.
objsubid	smallint	Column number for a column in the table (**0** if the target is of other object type and **NULL** if the object is not a general database object).
virtualtransaction	text	Virtual ID of the transaction holding or awaiting this lock.
pid	bigint	Logical ID of the server thread holding or awaiting this lock. The value is **NULL** if the lock is held by a prepared transaction.
mode	text	Lock mode held or desired by this thread.
granted	Boolean	• The value is **TRUE** if the lock is a held lock. • The value is **FALSE** if the lock is an awaited lock.
fastpath	Boolean	The value is **TRUE** if the lock is obtained through **fast-path**, and is **FALSE** if the lock is obtained through the main lock table.

13.2.13 Wait Events

13.2.13.1 WAIT_EVENTS

WAIT_EVENTS displays statistics about wait events on the current node, as described in **Table 13-160**. For details about the events, see **Table 12-412**, **Table**

12-413, Table 12-414, and Table 12-415. For details about the impact of each transaction lock on services, see LOCK.

Table 13-160 WAIT_EVENTS columns

Name	Type	Description
nodename	text	Database process name.
type	text	Event type.
event	text	Event name.
wait	bigint	Number of waiting times.
failed_wait	bigint	Number of waiting failures.
total_wait_time	bigint	Total waiting time (unit: μs).
avg_wait_time	bigint	Average waiting time (unit: μs).
max_wait_time	bigint	Maximum waiting time (unit: μs).
min_wait_time	bigint	Minimum waiting time (unit: μs).
last_updated	timestamp with time zone	Last time when the event was updated.

13.2.13.2 GLOBAL_WAIT_EVENTS

GLOBAL_WAIT_EVENTS displays statistics about wait events on each node, as described in Table 13-161.

Table 13-161 GLOBAL_WAIT_EVENTS columns

Name	Type	Description
nodename	text	Database process name
type	text	Event type
event	text	Event name
wait	bigint	Number of waiting times
failed_wait	bigint	Number of waiting failures
total_wait_time	bigint	Total waiting time (unit: μs)
avg_wait_time	bigint	Average waiting time (unit: μs)
max_wait_time	bigint	Maximum waiting time (unit: μs)

Name	Type	Description
min_wait_time	bigint	Minimum waiting time (unit: μs)
last_updated	timestamp with time zone	Last time when the event was updated

13.2.14 Configuration

13.2.14.1 CONFIG_SETTINGS

CONFIG_SETTINGS displays information about parameters of the running database, as described in **Table 13-162**.

Table 13-162 CONFIG_SETTINGS columns

Name	Type	Description
name	text	Parameter name
setting	text	Current parameter value
unit	text	Implicit unit of the parameter
category	text	Logical group of the parameter
short_desc	text	Brief description of the parameter
extra_desc	text	Detailed description of the parameter
context	text	Context required to set the parameter value, including **internal**, **postmaster**, **sighup**, **backend**, **superuser**, and **user**.
vartype	text	Parameter type, including **bool**, **enum**, **integer**, **real**, or **string**.
source	text	Method of assigning the parameter value
min_val	text	Maximum value of the parameter. If the parameter type is not numeric, the value of this column is **null**.
max_val	text	Minimum value of the parameter. If the parameter type is not numeric, the value of this column is **null**.
enumvals	text[]	Valid values of an enum-type parameter. If the parameter type is not enum, the value of this column is **null**.

Name	Type	Description
boot_val	text	Default parameter value used upon the database startup
reset_val	text	Default parameter value used upon the database reset
sourcefile	text	Configuration file used to set parameter values. If parameter values are not configured using the configuration file, the value of this column is **null**.
sourceline	integer	Row number in the configuration file for setting parameter values. If parameter values are not configured using the configuration file, the value of this column is **null**.

13.2.14.2 GLOBAL_CONFIG_SETTINGS

GLOBAL_CONFIG_SETTINGS displays information about parameters of running databases on each node, as described in **Table 13-163**.

Table 13-163 GLOBAL_CONFIG_SETTINGS columns

Name	Type	Description
node_name	text	Node name
name	text	Parameter name
setting	text	Current parameter value
unit	text	Implicit unit of the parameter
category	text	Logical group of the parameter
short_desc	text	Brief description of the parameter
extra_desc	text	Detailed description of the parameter
context	text	Context required to set the parameter value, including **internal**, **postmaster**, **sighup**, **backend**, **superuser**, and **user**.
vartype	text	Parameter type, including **bool**, **enum**, **integer**, **real**, or **string**.
source	text	Method of assigning the parameter value
min_val	text	Minimum value of the parameter. If the parameter type is not numeric, the value of this column is **null**.

Name	Type	Description
max_val	text	Maximum value of the parameter. If the parameter type is not numeric, the value of this column is **null**.
enumvals	text[]	Valid values of an enum-type parameter. If the parameter type is not enum, the value of this column is **null**.
boot_val	text	Default parameter value used upon the database startup
reset_val	text	Default parameter value used upon the database reset
sourcefile	text	Configuration file used to set parameter values. If parameter values are not configured using the configuration file, the value of this column is **null**.
sourceline	integer	Row number in the configuration file for setting parameter values. If parameter values are not configured using the configuration file, the value of this column is **null**.

13.2.15 Operator

13.2.15.1 OPERATOR_HISTORY_TABLE

OPERATOR_HISTORY_TABLE displays records about operators of completed jobs, as described in **Table 13-164**. Data is dumped from the kernel to this system catalog.

Table 13-164 OPERATOR_HISTORY_TABLE columns

Name	Type	Description
queryid	bigint	Internal query ID used for statement execution.
pid	bigint	Thread ID of the backend.
plan_node_id	integer	Plan node ID of the execution plan.
plan_node_name	text	Name of the operator corresponding to the plan node ID.
start_time	timestamp with time zone	Time when the operator starts to process the first data record.

Name	Type	Description
duration	bigint	Total execution time of the operator (unit: ms).
query_dop	integer	DOP of the operator.
estimated_rows	bigint	Number of rows estimated by the optimizer.
tuple_processed	bigint	Number of elements returned by the operator.
min_peak_memory	integer	Minimum peak memory used by the operator on database nodes (unit: MB).
max_peak_memory	integer	Maximum peak memory used by the operator on database nodes (unit: MB).
average_peak_memory	integer	Average peak memory used by the operator on database nodes (unit: MB).
memory_skew_percent	integer	Memory usage skew of the operator among database nodes.
min_spill_size	integer	Minimum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**).
max_spill_size	integer	Maximum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**).
average_spill_size	integer	Average spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**).
spill_skew_percent	integer	Database node spill skew when a spill occurs.
min_cpu_time	bigint	Minimum execution time of the operator on database nodes (unit: ms).
max_cpu_time	bigint	Maximum execution time of the operator on database nodes (unit: ms).
total_cpu_time	bigint	Total execution time of the operator on database nodes (unit: ms).
cpu_skew_percent	integer	Execution time skew among database nodes.

Name	Type	Description
warning	text	Warning. The following warnings are displayed: • Sort/SetOp/HashAgg/HashJoin spill • Spill file size large than 256MB • Broadcast size large than 100MB • Early spill • Spill times is greater than 3 • Spill on memory adaptive • Hash table conflict

13.2.15.2 OPERATOR_HISTORY

OPERATOR_HISTORY displays records of operators in jobs that have been executed by the current user on the current primary database node.

13.2.15.3 OPERATOR_RUNTIME

OPERATOR_RUNTIME displays information about operators of the jobs that are being executed by the current user, as described in **Table 13-165**.

Table 13-165 OPERATOR_RUNTIME columns

Name	Type	Description
queryid	bigint	Internal query ID used for statement execution
pid	bigint	Thread ID of the backend
plan_node_id	integer	Plan node ID of the execution plan of a query
plan_node_name	text	Name of the operator corresponding to the plan node ID
start_time	timestamp with time zone	Time when the operator starts to process the first data record
duration	bigint	Total execution time of the operator, in ms
status	text	Execution status of the current operator, which can be **finished** or **running**.
query_dop	integer	DOP of the operator
estimated_rows	bigint	Number of rows estimated by the optimizer
tuple_processed	bigint	Number of elements returned by the operator

Name	Type	Description
min_peak_memory	integer	Minimum peak memory used by the operator on database nodes (unit: MB)
max_peak_memory	integer	Maximum peak memory used by the operator on database nodes (unit: MB)
average_peak_memory	integer	Average peak memory used by the operator on database nodes (unit: MB)
memory_skew_percent	integer	Memory usage skew of the operator among database nodes
min_spill_size	integer	Minimum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
max_spill_size	integer	Maximum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
average_spill_size	integer	Average spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
spill_skew_percent	integer	Database node spill skew when a spill occurs
min_cpu_time	bigint	Minimum execution time of the operator on database nodes (unit: ms).
max_cpu_time	bigint	Maximum execution time of the operator on database nodes (unit: ms).
total_cpu_time	bigint	Total execution time of the operator on database nodes (unit: ms).
cpu_skew_percent	integer	Execution time skew among database nodes
warning	text	Warning. The following warnings are displayed: ● Sort/SetOp/HashAgg/HashJoin spill ● Spill file size large than 256MB ● Broadcast size large than 100MB ● Early spill ● Spill times is greater than 3 ● Spill on memory adaptive ● Hash table conflict

13.2.15.4 GLOBAL_OPERATOR_HISTORY

GLOBAL_OPERATOR_HISTORY displays records of operators in jobs that have been executed by the current user on the primary database node, as described in **Table 13-166**.

Table 13-166 GLOBAL_OPERATOR_HISTORY columns

Name	Type	Description
queryid	bigint	Internal query ID used for statement execution
pid	bigint	Thread ID of the backend
plan_node_id	integer	Plan node ID of the execution plan
plan_node_name	text	Name of the operator corresponding to the plan node ID
start_time	timestamp with time zone	Time when the operator starts to process the first data record
duration	bigint	Total execution time of the operator, in ms
query_dop	integer	DOP of the operator
estimated_rows	bigint	Number of rows estimated by the optimizer
tuple_processed	bigint	Number of elements returned by the operator
min_peak_memory	integer	Minimum peak memory used by the operator on database nodes (unit: MB)
max_peak_memory	integer	Maximum peak memory used by the operator on database nodes (unit: MB)
average_peak_memory	integer	Average peak memory used by the operator on database nodes (unit: MB)
memory_skew_percent	integer	Memory usage skew of the operator among database nodes
min_spill_size	integer	Minimum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
max_spill_size	integer	Maximum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
average_spill_size	integer	Average spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
spill_skew_percent	integer	Database node spill skew when a spill occurs

Name	Type	Description
min_cpu_time	bigint	Minimum execution time of the operator on database nodes (unit: ms)
max_cpu_time	bigint	Maximum execution time of the operator on database nodes (unit: ms)
total_cpu_time	bigint	Total execution time of the operator on database nodes (unit: ms)
cpu_skew_percent	integer	Execution time skew among database nodes
warning	text	Warning. The following warnings are displayed: 1. Sort/SetOp/HashAgg/HashJoin spill 2. Spill file size large than 256MB 3. Broadcast size large than 100MB 4. Early spill 5. Spill times is greater than 3 6. Spill on memory adaptive 7. Hash table conflict

13.2.15.5 GLOBAL_OPERATOR_HISTORY_TABLE

GLOBAL_OPERATOR_HISTORY_TABLE displays the records about operators of completed jobs on the primary database node. Data is dumped from the kernel to the system catalog **GS_WLM_OPERATOR_INFO**.
GLOBAL_OPERATOR_HISTORY_TABLE is a collection view for querying the system catalog **GS_WLM_OPERATOR_INFO** on the primary database node. Columns in this view are the same as those in **Table 13-166**.

13.2.15.6 GLOBAL_OPERATOR_RUNTIME

GLOBAL_OPERATOR_RUNTIME displays information about operators of the jobs that are being executed by the current user on the primary database node, as described in **Table 13-167**.

Table 13-167 GLOBAL_OPERATOR_RUNTIME columns

Name	Type	Description
queryid	bigint	Internal query ID used for statement execution
pid	bigint	Thread ID of the backend
plan_node_id	integer	Plan node ID of the execution plan
plan_node_name	text	Name of the operator corresponding to the plan node ID

Name	Type	Description
start_time	timestamp with time zone	Time when the operator starts to process the first data record
duration	bigint	Total execution time of the operator, in ms
status	text	Execution status of the current operator, which can be **finished** or **running**.
query_dop	integer	DOP of the operator
estimated_rows	bigint	Number of rows estimated by the optimizer
tuple_processed	bigint	Number of elements returned by the operator
min_peak_memory	integer	Minimum peak memory used by the operator on database nodes (unit: MB)
max_peak_memory	integer	Maximum peak memory used by the operator on database nodes (unit: MB)
average_peak_memory	integer	Average peak memory used by the operator on database nodes (unit: MB)
memory_skew_percent	integer	Memory usage skew of the operator among database nodes
min_spill_size	integer	Minimum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
max_spill_size	integer	Maximum spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
average_spill_size	integer	Average spilled data among database nodes when a spill occurs (unit: MB) (default value: **0**)
spill_skew_percent	integer	Database node spill skew when a spill occurs
min_cpu_time	bigint	Minimum execution time of the operator on database nodes (unit: ms)
max_cpu_time	bigint	Maximum execution time of the operator on database nodes (unit: ms)
total_cpu_time	bigint	Total execution time of the operator on database nodes (unit: ms)
cpu_skew_percent	integer	Execution time skew among database nodes

Name	Type	Description
warning	text	Warning. The following warnings are displayed: • Sort/SetOp/HashAgg/HashJoin spill • Spill file size large than 256MB • Broadcast size large than 100MB • Early spill • Spill times is greater than 3 • Spill on memory adaptive • Hash table conflict

13.2.16 Workload Manager

13.2.16.1 WLM_USER_RESOURCE_CONFIG

WLM_USER_RESOURCE_CONFIG displays the resource configuration information of a user, as described in **Table 13-168**.

Table 13-168 WLM_USER_RESOURCE_CONFIG columns

Name	Type	Description
userid	oid	OID of the user
username	name	Username
sysadmin	boolean	Whether the user has the **sysadmin** permission
rpoid	oid	OID of the resource pool
respool	name	Name of a resource pool
parentid	oid	OID of the parent user
totalspace	bigint	Size of the occupied space
spacelimit	bigint	Upper limit of the space size
childcount	integer	Number of child users
childlist	text	Child user list

13.2.16.2 WLM_USER_RESOURCE_RUNTIME

WLM_USER_RESOURCE_RUNTIME displays resource usage of all users. Only administrators can query this view, as described in **Table 13-169**. This view is valid only when the GUC parameter **use_workload_manager** is set to **on**.

Table 13-169 WLM_USER_RESOURCE_RUNTIME columns

Name	Type	Description
username	name	Username
used_memory	integer	Used memory (unit: MB)
total_memory	integer	Available memory (unit: MB) The value **0** indicates that the available memory is not limited and depends on the maximum memory available in the database.
used_cpu	integer	Number of CPU cores in use
total_cpu	integer	Total number of CPU cores of the Cgroup associated with the user on the node
used_space	bigint	Used storage space (unit: KB)
total_space	bigint	Available storage space (unit: KB) The value **-1** indicates that the maximum storage space is not limited.
used_temp_space	bigint	Used temporary space (reserved column; unit: KB)
total_temp_space	bigint	Available temporary storage space (reserved column; unit: KB) The value **-1** indicates that the maximum temporary storage space is not limited.
used_spill_space	bigint	Used space for storing spilled data (reserved column; unit: KB)
total_spill_space	bigint	Available storage space for spilled data (reserved column; unit: KB) The value **-1** indicates that the maximum space for spilled data is not limited.

13.2.17 Global Plancache

Global plan cache (GPC) views are valid only when **enable_global_plancache** is set to **on**.

13.2.17.1 GLOBAL_PLANCACHE_STATUS

GLOBAL_PLANCACHE_STATUS displays the GPC status information, as described in **Table 13-170**.

Table 13-170 GLOBAL_PLANCACHE_STATUS columns

Name	Type	Description
nodename	text	Name of the node that the plan cache belongs to
query	text	Text of query statements
refcount	integer	Number of times that the plan cache is referenced
valid	bool	Whether the plan cache is valid
databaseid	oid	ID of the database that the plan cache belongs to
schema_name	text	Schema that the plan cache belongs to
params_num	integer	Number of parameters
func_id	oid	OID of the stored procedure where the plan cache is located. If the plancache does not belong to the stored procedure, the value is **0**.
pkg_id	oid	Package to which the stored procedure where the plancache is located belongs. If the stored procedure does not belong to a package, the value is **0**.
stmt_id	integer	Sequence number of the statement plan in the stored procedure.

13.2.17.2 GLOBAL_PLANCACHE_CLEAN

GLOBAL_PLANCACHE_CLEAN clears the global plan cache that is not used on all nodes. The return value is of the Boolean type.

13.2.18 RTO & RPO

13.2.18.1 global_rto_status

global_rto_status displays log flow control information about the primary and standby nodes (except the current node and standby DNs), as described in **Table 13-171**.

Table 13-171 global_rto_status columns

Parameter	Type	Description
node_name	text	Node name (including the primary and standby nodes)

Parameter	Type	Description
rto_info	text	Flow control information, including the current log flow control time (unit: second) of the standby node, the expected flow control time (unit: second) specified by the GUC parameter, and the primary node sleep time (unit: μs) required to reach the expectation

13.2.18.2 global_streaming_hadr_rto_and_rpo_stat

global_streaming_hadr_rto_and_rpo_stat displays the log flow control information about the primary and standby database instances in the streaming DR scenario, as described in **Table 13-172**. (This schema can be used only on the primary DN of the primary database instance. Statistics cannot be obtained from the standby DN or the standby database instance.)

Table 13-172 Parameters

Parameter	Type	Description
hadr_sender_node_name	text	Node name, including the primary database instance and the first standby node of the standby database instance.
hadr_receiver_node_name	text	Name of the first standby node of the standby database instance.
current_rto	int	Flow control information, that is, log RTO time of the current primary and standby database instances (unit: second). **-1** indicates that the majority of the DR cluster is abnormal.
target_rto	int	Flow control information, that is, the RTO time between the target primary and standby database instances (unit: second).
current_rpo	int	Flow control information, that is, log RPO time of the current primary and standby database instances (unit: second). **-1** indicates that the majority of the DR cluster is abnormal.
target_rpo	int	Flow control information, that is, the RPO time between the target primary and standby database instances (unit: second).
rto_sleep_time	int	RTO flow control information, that is, the expected sleep time (unit: μs) required by walsender on the host to reach the specified RTO.
rpo_sleep_time	int	RPO flow control information, that is, the expected sleep time (unit: μs) required by xlogInsert on the host to reach the specified RPO.

13.2.19 AI Watchdog

13.2.19.1 ai_watchdog_monitor_status

Table 13-173 ai_watchdog_monitor_status parameters

Parameter	Type	Description
metric_name	text	Metric names: • **tps**: TPS. • **tps_hourly**: average TPS per hour. • **shared_used_mem**: the used shared memory (MB). • **dynamic_used_shrctx**: the used shared memory context (MB). • **other_used_mem**: other used memory (MB). • **process_used_mem**: the used resident memory (MB). • **dynamic_used_mem**: the used dynamic memory (MB). • **malloc_failures**: the number of memory allocation failures in each collection interval. • **D_state_rate**: percentage of threads in the D state. • **R_state_rate**: percentage of threads in the R state. • **S_state_rate**: ratio of threads in the S state. • **db_state**: database state (**68** indicates D, **82** indicates R, and **83** indicates S). • **cpu_usage**: CPU usage. The upper limit is 100. • **disk_io**: disk I/O delay between two collection intervals. • **network_io**: network I/O delay between two collection intervals. • **threadpool_usage**: thread pool usage. • **threadpool_hang_rate**: percentage of the thread pool group in the hang state.
max_length	int	Collection queue length.
current_length	int	The number of currently collected samples.

Parameter	Type	Description
collection_interval	int	Collection interval, in seconds.
latest_value	int	Value collected last time. If no value is collected, the value is **null**.
last_report	timestamp	The last collection time.

13.2.19.2 ai_watchdog_detection_warnings

Table 13-174 ai_watchdog_detection_warnings parameters

Parameter	Type	Description
event	text	Event name.
cause	text	Event cause.
details	text	Event details.
time	timestamp	Reporting time.
need_to_handle	bool	Determines whether automatic processing is required.

13.2.19.3 ai_watchdog_parameters

Table 13-175 ai_watchdog_parameters parameters

Parameter	Type	Description
name	text	Parameter name. The options are as follows: • **enable_ai_watchdog**: determines whether to enable this function. • **ai_watchdog_max_consuming_time_ms**: specifies the maximum duration. • **ai_watchdog_used_memory_kb**: specifies the memory used by this function. • **ai_watchdog_detection_times**: specifies the number of detection times. • **enable_self_healing**: determines whether self-healing can be performed after a problem is detected. • **oom_detected_times**: specifies the number of detected OOMs. • **hang_detected_times**: specifies the number of detected hang times. • **enable_oom_detection**: determines whether the OOM detection function is automatically enabled. • **in_wait_time**: determines whether the system is waiting. • **other_used_memory_has_risk**: determines whether other memory usage is risky. • **shared_used_mem_has_risk**: determines whether risks exist when the shared memory context is used. • **dynamic_used_shrctx_has_risk**: determines whether the dynamic memory usage is risky.
value	text	Parameter value

13.2.19.4 ai_watchdog_ftask_status

Table 13-176 ai_watchdog_ftask_status parameters

Parameter	Type	Description
name	text	Name of the watchdog feeding task.
timeout_threshold	int	Timeout threshold of the watchdog feeding task.
register_time	timestamp	Registration time of the watchdog feeding task.
thread_id	bigint	ID of the thread on which the watchdog feeding task is used.
thread_name	text	Name of the thread on which the watchdog feeding task is used.
last_feed	timestamp	Latest time when the watchdog is fed. The default initial value is **register_time**.
restart_count	int	Number of times that the same name of watchdog feeding task is repeatedly registered due to the same backend thread exception. This parameter is cleared after the task invokes the interface for canceling watchdog resetting task registration.
timeout_action	text	Timeout processing action specified during task registration. Currently, the following actions are provided: • **WATCHDOG_REPORT_WARNING**: Prompt the timeout locating and demarcation. • **WATCHDOG_SEND_SIGTERM**: Send SIGTERM signals to threads. • **WATCHDOG_SEND_SIGUSR2**: Send SIGUSR2 signals to threads. • **WATCHDOG_SEND_SIGQUIT**: Send SIGQUIT signals to threads. • **WATCHDOG_EXIT_PROCESS**: Exit the process.

13.3 WDR Snapshot Schema

After the WDR snapshot function is enabled (the GUC parameter **enable_wdr_snapshot** is set to **on**), objects are created in the snapshot schema of user tablespace **pg_default** and database postgres to flush WDR snapshot data. By default, the initial user or the **monadmin** user can access and operate the object of the snapshot schema.

You can set the GUC parameter **wdr_snapshot_retention_days** to automatically manage the snapshot lifecycle.

NOTICE

Do not add, delete, or modify tables in the snapshot schema. Manual modification or damage to these tables may cause WDR exceptions or even WDR unavailability.

13.3.1 Original Information of WDR Snapshots

13.3.1.1 SNAPSHOT.SNAPSHOT

SNAPSHOT records the index information, start time, and end time of WDR snapshots stored in the current system. The result can be queried only in the system database but cannot be queried in the user database, as described in **Table 13-177**. After the WDR snapshot function is enabled (the GUC parameter **enable_wdr_snapshot** is set to **on**), the table is created.

Table 13-177 SNAPSHOT attributes

Name	Type	Description	Example
snapshot_id	bigint	WDR snapshot ID.	1
start_ts	timestamp	Start time of a WDR snapshot.	2019-12-28 17:11:27.423742+08
end_ts	timestamp	End time of a WDR snapshot.	2019-12-28 17:11:43.67726+08
version	int	Version of a WDR snapshot.	1
snap_flag	bigint	Attribute of the WDR snapshot. • **NULL**: The snapshot is of an earlier version. • **0**: The snapshot is a full snapshot. • **1**: The snapshot is an incremental snapshot.	0
base_snapshot_id	bigint	ID of the full snapshot corresponding to the WDR snapshot.	1

13.3.1.2 SNAPSHOT.TABLES_SNAP_TIMESTAMP

TABLES_SNAP_TIMESTAMP records the databases, table objects, and start and end time of all WDR snapshots, as described in **Table 13-178**. After the WDR snapshot function is enabled (the GUC parameter **enable_wdr_snapshot** is set to **on**), the table is created.

Table 13-178 TABLES_SNAP_TIMESTAMP attributes

Name	Type	Description	Example
snapshot_id	bigint	WDR snapshot ID.	1
node_name	name	Name of the node to which the table information of the WDR snapshot belongs.	dn_6001
db_name	text	Database corresponding to a WDR snapshot.	tpcc1000
tablename	text	Table corresponding to a WDR snapshot.	snap_xc_statio_all_indexes
start_ts	timestamp	Start time of a WDR snapshot.	2019-12-28 17:11:27.425849+08
end_ts	timestamp	End time of a WDR snapshot.	2019-12-28 17:11:27.707398+08

13.3.1.3 SNAP_SEQ

SNAP_SEQ is an incremental sequence that provides the snapshot IDs for the WDR snapshot. After the WDR snapshot function is enabled (the GUC parameter **enable_wdr_snapshot** is set to **on**), the sequence is created.

13.3.2 WDR Snapshot Data Table

The naming rule of a WDR Snapshot data table is **snap**_*{Source data table}*.

The source of WDR snapshot data tables is the view under **DBE_PERF Schema**. After the WDR snapshot function is enabled (the GUC parameter **enable_wdr_snapshot** is set to **on**), the WDR snapshot data table is created.

13.4 DBE_PLDEBUGGER Schema

DBE_PLDEBUGGER Schema system functions are used to debug stored procedures. This chapter describes the interfaces supported by DBE_PLDEBUGGER Schema.

Only an administrator has the permission to execute these debugging interfaces, but does not have the permission to modify or create functions in the schema. Common users can debug only non-system functions in the public schema or user-created schema. Common users are not allowed to debug system functions.

NOTICE

- When a user is created in the function body, the plaintext password is returned when attach, next, continue, info_code, step, info_breakpoint, backtrace or finish is called. You are advised not to create a user in the function body.
- During the debugging of a stored procedure, if the stored procedure to be debugged involves lock operations, do not perform operations that may cause deadlocks on the debugging end.
- The debugging end and the debugged end must be connected to the same database. Otherwise, the API that uses the function OID to obtain function information is unavailable.

The administrator can run the following command to grant the **gs_role_pldebugger** role and debugger permissions to a user:

```
GRANT gs_role_pldebugger to user;
```

Two clients are required to connect to the database. One client is responsible for executing the debugging API as the debug end, and the other client is responsible for executing the debugging function to control the execution of stored procedures on the server. The following is an example:

- Prepare for debugging.

  Use PG_PROC to find the OID of the stored procedure to be debugged and execute DBE_PLDEBUGGER.turn_on(oid). In this case, the client functions as the server.

  ```
  gaussdb=# CREATE OR REPLACE PROCEDURE test_debug ( IN  x INT)
  AS
  BEGIN
      INSERT INTO t1 (a) VALUES (x);
      DELETE FROM t1 WHERE a = x;
  END;
  /
  CREATE PROCEDURE
  gaussdb=# SELECT OID FROM PG_PROC WHERE PRONAME='test_debug';
    oid
  -------
   16389
  (1 row)
  gaussdb=# SELECT * FROM DBE_PLDEBUGGER.turn_on(16389);
   nodename | port
  ----------+------
   datanode |    0
  (1 row)
  ```

- Start debugging.

  When the server executes the stored procedure, the server hangs before the first SQL statement in the stored procedure and waits for the debugging message sent by the debug end. Debugging is supported only by directly executing a stored procedure and cannot be achieved by invoking an executed stored procedure through a trigger.

  ```
  gaussdb=# call test_debug(1);
  ```

  Start another client as the debug end and invoke **DBE_PLDEBUGGER.attach** to attach with the stored procedure for debugging based on the data returned by **turn_on**.

  ```
  gaussdb=# SELECT * FROM DBE_PLDEBUGGER.attach('datanode',0);
   funcoid |  funcname  | lineno |              query
  ```



```
---------+------------+--------+---------------------------------
  16389 | test_debug |      3 |   INSERT INTO t1 (a) VALUES (x);
(1 row)
```

Execute the next statement on the client where the attach operation is performed.

```
gaussdb=# SELECT * FROM DBE_PLDEBUGGER.next();
 funcoid |  funcname  | lineno |        query
---------+------------+--------+----------------------
  16389 | test_debug |      0 | [EXECUTION FINISHED]
(1 row)
```

Execute the following variable operations on the client where the attach command is performed.

```
gaussdb=# SELECT * FROM DBE_PLDEBUGGER.info_locals(); -- Print all variables.
 varname | vartype | value | package_name | isconst
---------+---------+-------+--------------+---------
 x       | int4    | 1     |              | f
(1 row)
gaussdb=# SELECT * FROM DBE_PLDEBUGGER.set_var('x', 2); -- Assign a value to a variable.
 set_var
---------
 t
(1 row)
gaussdb=# SELECT * FROM DBE_PLDEBUGGER.print_var('x'); -- Print a single variable.
 varname | vartype | value | package_name | isconst
---------+---------+-------+--------------+---------
 x       | int4    | 2     |              | f
(1 row)
```

Directly execute the stored procedure that is being debugged.

```
gaussdb=# SELECT * FROM DBE_PLDEBUGGER.continue();
 funcoid |  funcname  | lineno |        query
---------+------------+--------+----------------------
  16389 | test_debug |      0 | [EXECUTION FINISHED]
(1 row)
```

When the stored procedure reports an error, the following information is displayed. In this case, the suspended process logic is triggered.

```
gaussdb=# SELECT * FROM DBE_PLDEBUGGER.continue();
 funcoid |  funcname  | lineno |            query
---------+------------+--------+-------------------------------
  16389 | test_debug |      0 | [EXECUTION HAS ERROR OCCURRED!]
(1 row)
```

If the suspended process logic is triggered, you can call the error_info_locals,error_backtrace, error_end, and print_var APIs to view information. Other APIs cannot be used anymore. You need to use error_end to end the suspended process logic.

```
gaussdb=# SELECT * FROM DBE_PLDEBUGGER.error_end();
 funcoid |  funcname  | lineno |      query
---------+------------+--------+------------------
  16389 | test_debug |      0 | [END HANG ERROR!]
(1 row)
```

Exit the stored procedure that is being debugged (the suspended process logic is unavailable), and do not execute statements that have not been executed before.

```
gaussdb=# SELECT * FROM DBE_PLDEBUGGER.abort();
 abort
-------
 t
(1 row)
```

View the code information on the client and identify the line number of the breakpoint that can be set.


```
gaussdb=# SELECT * FROM DBE_PLDEBUGGER.info_code(16389);
 lineno |                          query                          | canbreak
--------+---------------------------------------------------------+----------
        | CREATE OR REPLACE PROCEDURE public.test_debug( IN  x INT) | f
      1 | AS  DECLARE                                               | f
      2 | BEGIN                                                     | f
      3 |     INSERT INTO t1 (a) VALUES (x);                        | t
      4 |     DELETE FROM t1 WHERE a = x;                           | t
      5 | END;                                                      | f
      6 | /                                                         | f
(7 rows)
```

Sets a breakpoint.

```
gaussdb=# SELECT * FROM DBE_PLDEBUGGER.add_breakpoint(16389,4);
breakpointno
--------------
           0
(1 row)
```

View the breakpoint information.

```
gaussdb=# SELECT * FROM DBE_PLDEBUGGER.info_breakpoints();
 breakpointno | funcoid | lineno |              query              | enable
--------------+---------+--------+---------------------------------+--------
            0 |   16389 |      4 |     DELETE FROM t1 WHERE a = x; | t
(1 row)
```

Execute to the breakpoint.

```
gaussdb=# SELECT * FROM DBE_PLDEBUGGER.continue();
 funcoid |  funcname  | lineno |              query
---------+------------+--------+---------------------------------
   16389 | test_debug |      4 |     DELETE FROM t1 WHERE a = x;
(1 row)
```

After the stored procedure is executed, the debugging automatically exits. To debug the stored procedure again, you need to attach again. If the server does not need to be debugged, run the **turn_off** command to disable the debugging or exit the session. **Table 13-179** describes the debugging interfaces.

Table 13-179 DBE_PLDEBUGGER

Interface	Description
DBE_PLDEBUGGER.turn_on	Invoked by the server, indicating that the stored procedure can be debugged. After the interface is invoked, the stored procedure is hung to wait for debugging information.
DBE_PLDEBUGGER.turn_off	Invoked by the server, indicating that debugging the stored procedure is disabled.
DBE_PLDEBUGGER.local_debug_server_info	Invoked by the server to print all stored procedures that have been turned on in the current session.
DBE_PLDEBUGGER.attach	Invoked by the debug end to attach with the stored procedure that is being debugged.
DBE_PLDEBUGGER.info_locals	Invoked by the debug end to print the current values of variables in the stored procedure that is being debugged.

Interface	Description
DBE_PLDEBUGGER.next	Invoked by the debug end to execute the next step.
DBE_PLDEBUGGER.continue	Invoked by the debug end to continue the execution until the breakpoint or stored procedure ends.
DBE_PLDEBUGGER.abort	Invoked by the debug end to stop debugging. The server reports a long jump error.
DBE_PLDEBUGGER.print_var	Invoked by the debug end to print the current values of specified variables in the stored procedure that is being debugged.
DBE_PLDEBUGGER.info_code	Invoked by the debug end or server to print the source statement of a specified stored procedure and the line number corresponding to each line.
DBE_PLDEBUGGER.step	Invoked by the debug end to execute step by step.
DBE_PLDEBUGGER.add_breakpoint	Invoked by the debug end to add a breakpoint.
DBE_PLDEBUGGER.delete_breakpoint	Invoked by the debug end to delete a breakpoint.
DBE_PLDEBUGGER.info_breakpoints	Invoked by the debug end to view all breakpoints.
DBE_PLDEBUGGER.backtrace	Invoked by the debug end to check the current call stack.
DBE_PLDEBUGGER.enable_breakpoint	Invoked by the debug end to enable breakpoints.
DBE_PLDEBUGGER.disable_breakpoint	Invoked by the debug end to disable breakpoints.
DBE_PLDEBUGGER.finish	Invoked by the debug end to continue the debugging until the breakpoint is reached or the upper-layer call stack is returned.
DBE_PLDEBUGGER.set_var	Invoked by the debug end to assign a value to a variable.

13.4.1 DBE_PLDEBUGGER.turn_on

This function is used to mark a stored procedure as debuggable, as described in Table 13-180. After **turn_on** is executed, the server can execute the stored procedure for debugging. You need to manually obtain the OID of the stored procedure based on the PG_PROC system catalog and transfer it to the function. After **turn_on** is executed, the execution of the stored procedure in the current session is hung before the first SQL statement to wait for the debugging

instruction from the debug end. This setting is cleared by default after the session is disconnected. Currently, stored procedures and functions with autonomous transactions enabled cannot be debugged. After **turn_on** is executed, the number of stored procedures cannot exceed 100.

The function prototype is as follows:

```
DBE_PLDEBUGGER.turn_on(Oid)
RETURN Record;
```

Table 13-180 turn_on input parameters and return values

Name	Type	Description
func_oid	IN oid	Function OID.
nodename	OUT text	Node name.
port	OUT integer	Connection port number.

13.4.2 DBE_PLDEBUGGER.turn_off

This function is used only to remove the debugging flag added by **turn_on** to the current session. The return value indicates success or failure, as described in **Table 13-181**. You can run the **DBE_PLDEBUGGER.local_debug_server_info** command to query the OID of the stored procedure that has been turned on.

The function prototype is as follows:

```
DBE_PLDEBUGGER.turn_off(Oid)
RETURN boolean;
```

Table 13-181 turn_off input parameters and return values

Name	Type	Description
func_oid	IN oid	Function OID.
turn_off	OUT boolean	Whether turn-off is successful.

13.4.3 DBE_PLDEBUGGER.local_debug_server_info

This function is used to query the OID of the stored procedure that has been turned on in the current connection. You can use **funcoid** and **pg_proc** together to determine which stored procedures are to be debugged, as described in **Table 13-182**.

Table 13-182 local_debug_server_info returned input parameters and return values

Name	Type	Description
nodename	OUT text	Node name.
port	OUT bigint	Port number.
funcoid	OUT oid	Stored procedure OID.

13.4.4 DBE_PLDEBUGGER.attach

When the server executes a stored procedure, the server hangs the execution before the first statement and waits for attaching with the debug end. The debug end invokes the attach function and transfers node name and port number to attach with the specified stored procedure, as described in **Table 13-183**.

If an error is reported during debugging, the attach operation automatically becomes invalid. If the debug end is attached to another stored procedure during debugging, the debugging of the attached stored procedure becomes invalid. If the attach operation is performed repeatedly, the current stored procedure is disconnected.

Table 13-183 attach input parameters and return values

Name	Type	Description
nodename	IN text	Node name.
port	IN integer	Connection port number.
funcoid	OUT oid	Function ID.
funcname	OUT text	Function name.
lineno	OUT integer	Number of the next line in the current debugging process.
query	OUT text	Source code of the next line of the function that is being debugged.

13.4.5 DBE_PLDEBUGGER.info_locals

During debugging on the debug end, **info_locals** is invoked to print the variables in the current stored procedure. The input parameter **frameno** of this function indicates the stack layer to be traversed. This function can be called without **frameno**. By default, the top-layer stack variable is queried, as described in **Table 13-184**.

Table 13-184 info_locals input parameters and return values

Name	Type	Description
frameno	IN integer (optional)	Specified stack layer. The default value is the top layer.
varname	OUT text	Variable name.
vartype	OUT text	Variable type.
value	OUT text	Variable value.
package_name	OUT text	Name of the package corresponding to the variable. If the variable is not a package, the value is null.
isconst	OUT boolean	Whether the value is a constant.

13.4.6 DBE_PLDEBUGGER.next

This function is used to execute the current SQL statement in a stored procedure and return the number of the next SQL statement and the corresponding query, as described in **Table 13-185**.

Table 13-185 next input parameters and return values

Name	Type	Description
funcoid	OUT oid	Function ID.
funcname	OUT text	Function name.
lineno	OUT integer	Number of the next line in the current debugging process.
query	OUT text	Source code of the next line of the function that is being debugged.

13.4.7 DBE_PLDEBUGGER.continue

Executes the current stored procedure until reaching the next breakpoint or end, and returns the line number of the next execution and the corresponding query, as described in **Table 13-186**.

The function prototype is as follows:


```
DBE_PLDEBUGGER.continue()
RETURN Record;
```

Table 13-186 continue input parameters and return values

Name	Type	Description
funcoid	OUT oid	Function ID.
funcname	OUT text	Function name.
lineno	OUT integer	Number of the next line in the current debugging process.
query	OUT text	Source code of the next line of the function that is being debugged.

13.4.8 DBE_PLDEBUGGER.abort

This function is used to abort the stored procedure executed on the server and report an error. The return value indicates whether the abort message is successfully sent, as described in **Table 13-187**.

The function prototype is as follows:

```
DBE_PLDEBUGGER.abort()
RETURN boolean;
```

Table 13-187 abort input parameters and return values

Name	Type	Description
abort	OUT boolean	Success or failure.

13.4.9 DBE_PLDEBUGGER.print_var

During debugging on the debug end, print_var is called to print the name and value of the specified variable in the current stored procedure. You can also use this function to print variables when an error is reported, as described in **Table 13-188**. The input parameter **frameno** of this function indicates the stack layer to be traversed. This function can be invoked without **frameno**. By default, the top-layer stack variable is queried.

Table 13-188 print_var input parameters and return values

Name	Type	Description
var_name	IN text	Variable.

Name	Type	Description
frameno	IN integer (optional)	Specified stack layer. The default value is the top layer.
varname	OUT text	Variable name.
vartype	OUT text	Variable type.
value	OUT text	Variable value.
package_name	OUT text	Package name corresponding to the variable. This parameter is reserved and is left empty currently.
isconst	OUT boolean	Whether the value is a constant.

13.4.10 DBE_PLDEBUGGER.info_code

During debugging on the debug end, **info_code** is invoked to view the source statement of the specified stored procedure and the line number corresponding to each line. The line number starts from the function body, and the line number in the function header is empty, as described in **Table 13-189**.

Table 13-189 info_code input parameters and return values

Name	Type	Description
funcoid	IN oid	Function ID.
lineno	OUT integer	Line number.
query	OUT text	Source statement.
canbreak	OUT bool	Whether the current line supports breakpoints.

13.4.11 DBE_PLDEBUGGER.step

During debugging on the debug end, if a stored procedure is being executed, the stored procedure continues to be executed and information such as the line number in the first line of the stored procedure is returned. If the executed object is not a stored procedure, the return is the same as that for **next**. After the SQL statement is executed, information such as the line number in the next line is returned, as described in **Table 13-190**.

Table 13-190 step input parameters and return values

Name	Type	Description
funcoid	OUT oid	Function ID.
funcname	OUT text	Function name.
lineno	OUT integer	Number of the next line in the current debugging process.
query	OUT text	Source code of the next line of the function that is being debugged.

13.4.12 DBE_PLDEBUGGER.add_breakpoint

During debugging on the debug end, call **add_breakpoint** to add a breakpoint, as described in **Table 13-191**. If **–1** is returned, the specified breakpoint is invalid. Determine a proper breakpoint position by referring to the **canbreak** column in **DBE_PLDEBUGGER.info_code**. If the outer function does not have a breakpoint before entering the inner function, a breakpoint cannot be added to the outer function when the inner function is executed.

Table 13-191 add_breakpoint input parameters and return values

Name	Type	Description
funcoid	IN text	Function ID.
lineno	IN integer	Line number.
breakpointno	OUT integer	Breakpoint number.

13.4.13 DBE_PLDEBUGGER.delete_breakpoint

During debugging on the debug end, call **delete_breakpoint** to delete the existing breakpoint, as described in **Table 13-192**.

Table 13-192 delete_breakpoint input parameters and return values

Name	Type	Description
breakpointno	IN integer	Breakpoint number.
result	OUT bool	Whether the request is successful.

13.4.14 DBE_PLDEBUGGER.info_breakpoints

During debugging on the debug end, call **info_breakpoints** to view the current function breakpoint, as described in **Table 13-193**.

Table 13-193 info_breakpoints input parameters and return values

Name	Type	Description
breakpointno	OUT integer	Breakpoint number.
funcoid	OUT oid	Function ID.
lineno	OUT integer	Line number.
query	OUT text	Breakpoint content.
enable	OUT boolean	Whether the value is valid.

13.4.15 DBE_PLDEBUGGER.backtrace

During debugging on the debug end, call **backtrace** to view the current call stack, as described in **Table 13-194**.

Table 13-194 backtrace input parameters and return values

Name	Type	Description
frameno	OUT integer	Call stack ID.
funcname	OUT text	Function name.
lineno	OUT integer	Line number.
query	OUT text	Breakpoint content.
funcoid	OUT oid	Function OID.

13.4.16 DBE_PLDEBUGGER.enable_breakpoint

During debugging on the debug end, calls enable_breakpoint to enable breakpoints, as described in **Table 13-195**.

Table 13-195 enable_breakpoint input parameters and return values

Name	Type	Description
breakpointno	IN integer	Breakpoint number.

Name	Type	Description
result	OUT bool	Whether the request is successful.

13.4.17 DBE_PLDEBUGGER.disable_breakpoint

During debugging on the debug end, call disable_breakpoint to disable breakpoints, as described in **Table 13-196**.

Table 13-196 disable_breakpoint input parameters and return values

Name	Type	Description
breakpointno	IN integer	Breakpoint number.
result	OUT bool	Whether the request is successful.

13.4.18 DBE_PLDEBUGGER.finish

Executes the current SQL statement in the stored procedure until the next breakpoint is triggered or the next line of the upper-layer stack is executed, as described in **Table 13-197**.

Table 13-197 finish input parameters and return values

Name	Type	Description
funcoid	OUT oid	Function ID.
funcname	OUT text	Function name.
lineno	OUT integer	Number of the next line in the current debugging process.
query	OUT text	Source code of the next line of the function that is being debugged.

13.4.19 DBE_PLDEBUGGER.set_var

Changes the variable on the top-layer stack in the specified debugging stored procedure to the value of the input parameter, as described in **Table 13-198**. If a stored procedure contains variables with the same name, set_var supports only the setting of the first variable value.

Table 13-198 set_var input parameters and return values

Name	Type	Description
var_name	IN text	Variable name.
value	IN text	New value.
result	OUT boolean	Whether the request is successful.

13.4.20 DBE_PLDEBUGGER.error_backtrace

If the server is suspended due to an error reported by the stored procedure, you can call error_backtrace on the debug end to view the current function call stack, as described in **Table 13-199**. (This function is used only for suspension after an error is reported.)

Table 13-199 error_backtrace return values

Name	Type	Description
frameno	OUT integer	Call stack ID.
funcname	OUT text	Function name.
lineno	OUT integer	Line number.
query	OUT text	Breakpoint content.
funcoid	OUT oid	Function OID.

13.4.21 DBE_PLDEBUGGER.error_end

If the server is suspended due to an error reported by the stored procedure, you can call error_end on the debug end to end the suspended process, end the debugging process, return the ended suspended process, and stop debugging, as described in **Table 13-200**. (This function is used only for suspension after an error is reported.)

Table 13-200 error_end return values

Name	Type	Description
funcoid	OUT oid	Function ID.
funcname	OUT text	Function name.
lineno	OUT integer	The next debugging line number (fixed value: 0).

Name	Type	Description
query	OUT text	Ends the suspended process and stops debugging.

13.4.22 DBE_PLDEBUGGER.error_info_locals

If the server is suspended due to an error reported by the stored procedure, you can call error_info_locals on the debug end to print the variables in the current stored procedure. The input parameter **frameno** of this function indicates the stack layer to be traversed. This function can be called without input parameters. By default, the top-layer stack variable is queried, as described in **Table 13-201**. (This function is used only for suspension after an error is reported.)

Table 13-201 error_info_locals return values

Name	Type	Description
frameno	IN integer (optional)	Specified stack layer. The default value is the top layer.
varname	OUT text	Variable name.
vartype	OUT text	Variable type.
value	OUT text	Variable value.
package_name	OUT text	Name of the package corresponding to the variable. If the variable is not a package, the value is null.
isconst	OUT boolean	Whether the value is a constant.

13.5 DB4AI Schema

The DB4AI schema is used to store and manage dataset versions in the AI feature. The schema holds a snapshot of the original view of the data tables, a change record for each data version, and management information for the version snapshot. This schema is intended for common users. In this schema, users can query the snapshot version information created by DB4AI.SNAPSHOT.

13.5.1 DB4AI.SNAPSHOT

DB4AI.SNAPSHOT records the snapshots stored by the current user through the DB4AI.SNAPSHOT feature, as described in **Table 13-202**.

Table 13-202 Attributes of the DB4AI.SNAPSHOT table

Name	Type	Description	Instance
id	bigint	ID of the current snapshot	1
parent_id	bigint	ID of the parent snapshot	0
matrix_id	bigint	Matrix ID of the snapshot in CSS mode. Otherwise, the value is **NULL**.	0
root_id	bigint	ID of the initial snapshot, which is constructed from the operated data using db4ai.create_snapshot().	0
schema	name	Schema for exporting the snapshot view	public
name	name	Snapshot name, including the version suffix	example0@1.1.0
owner	name	Name of the user who creates the snapshot	nw
commands	text[]	A complete list of SQL statements that record how to generate this snapshot from its root snapshot	{DELETE,"WHERE id > 7"}
comment	text	Snapshot description	inherits from @1.0.0
published	boolean	**TRUE**, only when the snapshot has been published	f
archived	boolean	**TRUE**, only when the snapshot has been archived	f
created	timestamp without time zone	Timestamp marking the snapshot creation date	2021-08-25 10:59:52.955604
row_count	bigint	Number of data rows in the snapshot	8

13.5.2 DB4AI.CREATE_SNAPSHOT

CREATE_SNAPSHOT creates snapshots for the DB4AI feature, as described in **Table 13-203**. You can invoke **CREATE SNAPSHOT** to implement this function.

Table 13-203 DB4AI.CREATE_SNAPSHOT input parameters and return values

Parameter	Type	Description
i_schema	IN NAME	Name of the schema storing snapshots. The default value is the current user or **PUBLIC**.
i_name	IN NAME	Snapshot name.
i_commands	IN TEXT[]	SQL commands for obtaining data.
i_vers	IN NAME	Version suffix.
i_comment	IN TEXT	Snapshot description.
res	OUT db4ai.snapshot_name	Result.

13.5.3 DB4AI.CREATE_SNAPSHOT_INTERNAL

CREATE_SNAPSHOT_INTERNAL is a built-in execution function of the DB4AI.CREATE_SNAPSHOT function, as described in **Table 13-204**. The function involves information verification and cannot be directly invoked.

Table 13-204 DB4AI.CREATE_SNAPSHOT_INTERNAL input parameters and return values

Parameter	Type	Description
s_id	IN BIGINT	Snapshot ID.
i_schema	IN NAME	Namespace storing the snapshot.
i_name	IN NAME	Snapshot name.
i_commands	IN TEXT[]	SQL commands for obtaining data.
i_comment	IN TEXT	Snapshot description.
i_owner	IN NAME	Snapshot owner.

13.5.4 DB4AI.PREPARE_SNAPSHOT

PREPARE_SNAPSHOT prepares data for model training and interprets snapshots for the DB4AI feature, as described in **Table 13-205**. A snapshot provides a complete sequence of data and documents that are changed by all applications. You can invoke **PREPARE SNAPSHOT** to implement this function.

Table 13-205 DB4AI.PREPARE_SNAPSHOT input parameters and return values

Parameter	Type	Description
i_schema	IN NAME	Name of the schema storing snapshots. The default value is the current user or **PUBLIC**.
i_parent	IN NAME	Parent snapshot name.
i_commands	IN TEXT[]	DDL and DML commands for modifying snapshots.
i_vers	IN NAME	Version suffix.
i_comment	IN TEXT	Data comment.
res	OUT db4ai.snapshot_name	Result.

13.5.5 DB4AI.PREPARE_SNAPSHOT_INTERNAL

PREPARE_SNAPSHOT_INTERNAL is a built-in execution function of the DB4AI.PREPARE_SNAPSHOT function, as described in **Table 13-206**. The function involves information verification and cannot be directly invoked.

Table 13-206 Input parameters of DB4AI.PREPARE_SNAPSHOT_INTERNAL

Parameter	Type	Description
s_id	IN BIGINT	Snapshot ID.
p_id	IN BIGINT	Parent snapshot ID.
m_id	IN BIGINT	Matrix ID.
r_id	IN BIGINT	Root snapshot ID.
i_schema	IN NAME	Snapshot schema.
i_name	IN NAME	Snapshot name.
i_commands	IN TEXT[]	DDL and DML commands for modifying snapshots.
i_comment	IN TEXT	Snapshot description.
i_owner	IN NAME	Snapshot owner.
i_idx	INOUT INT	Index of exec_cmds.
i_exec_cmds	INOUT TEXT[]	DDL and DML commands for execution.

Parameter	Type	Description
i_mapping	IN NAME[]	Maps user columns to backup columns. A rule is generated if the value is not **NULL**.

13.5.6 DB4AI.ARCHIVE_SNAPSHOT

ARCHIVE_SNAPSHOT archives snapshots for the DB4AI feature, as described in **Table 13-207**. You can invoke **ARCHIVE SNAPSHOT** to implement this function. After taking effect, the snapshot cannot be used for training.

Table 13-207 DB4AI.ARCHIVE_SNAPSHOT input parameters and return values

Parameter	Type	Description
i_schema	IN NAME	Name of the schema storing snapshots. The default value is the current user.
i_name	IN NAME	Snapshot name.
res	OUT db4ai.snapshot_name	Result.

13.5.7 DB4AI.PUBLISH_SNAPSHOT

PUBLISH_SNAPSHOT publishes snapshots for the DB4AI feature, as described in **Table 13-208**. You can invoke **PUBLISH SNAPSHOT** to implement this function.

Table 13-208 DB4AI.PUBLISH_SNAPSHOT input parameters and return values

Parameter	Type	Description
i_schema	IN NAME	Name of the schema storing snapshots. The default value is the current user or **PUBLIC**.
i_name	IN NAME	Snapshot name.
res	OUT db4ai.snapshot_name	Result.

13.5.8 DB4AI.MANAGE_SNAPSHOT_INTERNAL

MANAGE_SNAPSHOT_INTERNAL is a built-in execution function of the **DB4AI.PUBLISH_SNAPSHOT** and **DB4AI.ARCHIVE_SNAPSHOT** functions, as described in **Table 13-209**. The function involves information verification and cannot be directly invoked.

Table 13-209 DB4AI.MANAGE_SNAPSHOT_INTERNAL input parameters and return values

Parameter	Type	Description
i_schema	IN NAME	Name of the schema storing snapshots.
i_name	IN NAME	Snapshot name.
publish	IN BOOLEAN	Whether the snapshot is published.
res	OUT db4ai.snapshot_name	Result.

13.5.9 DB4AI.SAMPLE_SNAPSHOT

SAMPLE_SNAPSHOT samples basic data to generate snapshots for the DB4AI feature, as described in **Table 13-210**. You can invoke **SAMPLE SNAPSHOT** to implement this function.

Table 13-210 DB4AI.SAMPLE_SNAPSHOT input parameters and return values

Parameter	Type	Description
i_schema	IN NAME	Name of the schema storing snapshots.
i_parent	IN NAME	Parent snapshot name.
i_sample_infixes	IN NAME[]	Sample snapshot name infix.
i_sample_ratios	IN NUMBER[]	Size of each sample, which is used as the sample set ratio.
i_stratify	IN NAME[]	Layering strategy.
i_sample_comments	IN TEXT[]	Sample snapshot description.
res	OUT db4ai.snapshot_name	Result.

13.5.10 DB4AI.PURGE_SNAPSHOT

PURGE_SNAPSHOT deletes snapshots for the DB4AI feature, as described in **Table 13-211**. You can invoke **PURGE SNAPSHOT** to implement this function.

Table 13-211 DB4AI.PURGE_SNAPSHOT input parameters and return values

Parameter	Type	Description
i_schema	IN NAME	Name of the schema storing snapshots.
i_name	IN NAME	Snapshot name.
res	OUT db4ai.snapshot_name	Result.

13.5.11 DB4AI.PURGE_SNAPSHOT_INTERNAL

PURGE_SNAPSHOT_INTERNAL is a built-in execution function of the DB4AI.PURGE_SNAPSHOT function, as described in **Table 13-212**. The function involves information verification and cannot be directly invoked.

Table 13-212 DB4AI.PURGE_SNAPSHOT_INTERNAL input parameters and return values

Parameter	Type	Description
i_schema	IN NAME	Name of the schema storing snapshots.
i_name	IN NAME	Snapshot name.

13.6 DBE_PLDEVELOPER

The DBE_PLDEVELOPER system catalog records information required for compiling PL/SQL packages, functions, and stored procedures.

13.6.1 DBE_PLDEVELOPER.gs_source

Records compilation information about PL/SQL objects (stored procedures, functions, packages, and package bodies), as described in **Table 13-213**.

When the **plsql_show_all_error** parameter is enabled, information about PL/SQL object compilation success or failure is recorded in this table. When the **plsql_show_all_error** parameter is disabled, only information about correct compilation is inserted into this table.

CAUTION

1. The gs_source table records only user-defined original object statements. Even if a user uses ALTER to change the created schema or name, the information in the gs_source table does not change. If the user changes the schema or name of an object, the deleted object still exists in the gs_source table.
2. The owner in the gs_source table is the user who creates the table, not the user specified when the user creates the stored procedure or package.
3. By default, row-level security is not configured for the gs_source table in the database. If you want to use the database isolation feature, run the following statement to add row-level security:

```
ALTER TABLE dbe_pldeveloper.gs_source ENABLE ROW LEVEL SECURITY;
CREATE ROW LEVEL SECURITY POLICY all_data_rls ON dbe_pldeveloper.gs_source USING(owner =
(select oid from pg_roles where rolname=current_user));
```

Table 13-213 DBE_PLDEVELOPER.gs_source columns

Name	Type	Description
id	oid	Object ID.
owner	bigint	ID of the user who creates the object.
nspid	oid	Schema ID of an object.
name	name	Object name.
type	text	Object type (**procedure**/**function**/**package**/**package body**).
status	Boolean	Determines whether the creation is successful.
src	text	Original statement for creating an object.

13.6.2 DBE_PLDEVELOPER.gs_errors

Records errors that occur during PL/SQL object (stored procedure, function, package, and package body) compilation, as described in **Table 13-214**.

After the **plsql_show_all_error** parameter is enabled, if an error occurs during compilation, the error is skipped and the compilation continues, and the error information is recorded in **gs_errors**. If the **plsql_show_all_error** parameter is disabled, and the value of **behavior_compat_options** is not **skip_insert_gs_source**, then an error is reported and related information is inserted into this table.

The owner of the table is the user who creates the table. Modifying the owner of the stored procedure or package does not modify the table information.

CAUTION

1. If errors occur before AS/IS and after END when a packet header is created, the errors are not recorded in the **gs_errors** table. Instead, the error row numbers and content are directly returned on the client. The returned row numbers may be inaccurate. Some IS and END errors are not recorded in the **gs_errors** table.

2. If errors occur before AS/IS and after END when a packet body is created, the errors are not recorded in the **gs_errors** table. Instead, the error row numbers and content are directly returned on the client. The returned row numbers may be inaccurate. Some IS and END errors are not recorded in the **gs_errors** table.

3. If errors occur at the end of functions or stored procedures (after END) when a packet body is created, the errors are not recorded in the **gs_errors** table. Instead, the error row numbers and content are directly returned on the client. The row numbers may be inaccurate.

4. If errors occur at the beginning of functions or stored procedures (before AS/IS) when a packet body is created, the error row numbers are incorrect.

5. If a semicolon (;) is missing in the variable declaration when a packet header is created, the error is recorded in the **gs_errors** table but the row number is incorrect. If the parameter is enabled, the error is not recorded.

6. In a stored procedure or function in the package body, the error declared by the autonomous transaction identifier PRAGMA AUTONOMOUS_TRANSACTION may not be recorded in the **gs_errors** table.

7. The client directly reports an error, but the **gs_errors** table does not record the error. If the row number reported by the client is incorrect, this requirement does not correct the reported row number.

8. For an error in the middle of statements such as IF....THEN, FOR....LOOP, and WHEN.... THEN, or an EXCEPTION error, the error row number is in the current row instead of the row where the next semicolon is located.

9. When the BEGIN in the stored procedure or function in the package is incorrect, the error row number is incorrect.

Table 13-214 DBE_PLDEVELOPER.gs_errors columns

Name	Type	Description
id	oid	Object ID.
owner	bigint	ID of the user who creates the object.
nspid	oid	Schema ID of an object.
name	name	Object name.
type	text	Object type (**procedure**/**function**/**package**/**package body**).
line	integer	Line number.
src	text	Error message.

13.7 DBE_SQL_UTIL Schema

The DBE_SQL_UTIL schema stores tools for managing SQL patches, including creating, deleting, enabling, and disabling SQL patches. Common users have only the USAGE permission and do not have the CREATE, ALTER, DROP, and COMMENT permissions.

13.7.1 DBE_SQL_UTIL.create_hint_sql_patch

create_hint_sql_patch creates hint SQL patches and returns whether the execution is successful, as described in **Table 13-215**.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

Table 13-215 DBE_SQL_UTIL.create_hint_sql_patch input parameters and return values

Parameter	Type	Description
patch_name	IN name	Patch name.
unique_sql_id	IN bigint	Global unique ID.
hint_string	IN text	Hint text.
description	IN text	Patch description. The default value is **NULL**.
enabled	IN bool	Specifies whether the patch takes effect. The default value is **true**.
result	OUT bool	Specifies whether this operation is successful.

13.7.2 DBE_SQL_UTIL.create_abort_sql_patch

create_abort_sql_patch is an interface function used to create abort SQL patches. It returns whether the execution is successful, as described in **Table 13-216**.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

Table 13-216 DBE_SQL_UTIL.create_abort_sql_patch input parameters and return values

Parameter	Type	Description
patch_name	IN name	Patch name.

Parameter	Type	Description
unique_sql_id	IN bigint	Global unique ID.
description	IN text	Patch description. The default value is **NULL**.
enabled	IN bool	Specifies whether the patch takes effect. The default value is **true**.
result	OUT bool	Specifies whether this operation is successful.

13.7.3 DBE_SQL_UTIL.drop_sql_patch

drop_sql_patch is an interface function used to delete SQL patches. It returns whether the execution is successful, as described in **Table 13-217**.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

Table 13-217 DBE_SQL_UTIL.drop_sql_patch input parameters and return values

Parameter	Type	Description
patch_name	IN name	Patch name.
result	OUT bool	Specifies whether this operation is successful.

13.7.4 DBE_SQL_UTIL.enable_sql_patch

enable_sql_patch is an interface function used to enable SQL patches. It returns whether the execution is successful, as described in **Table 13-218**.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

Table 13-218 DBE_SQL_UTIL.enable_sql_patch input parameters and return values

Parameter	Type	Description
patch_name	IN name	Patch name.
result	OUT bool	Specifies whether this operation is successful.

13.7.5 DBE_SQL_UTIL.disable_sql_patch

disable_sql_patch is an interface function used to disable SQL patches. It returns whether the execution is successful, as described in **Table 13-219**.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

Table 13-219 DBE_SQL_UTIL.disable_sql_patch input parameters and return values

Parameter	Type	Description
patch_name	IN name	Patch name.
result	OUT bool	Specifies whether this operation is successful.

13.7.6 DBE_SQL_UTIL.show_sql_patch

show_sql_patch is an interface function used to display the SQL patch corresponding to a specified patch name and return the running result, as described in **Table 13-220**.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

Table 13-220 DBE_SQL_UTIL.show_sql_patch input parameters and return values

Parameter	Type	Description
patch_name	IN name	Patch name.
unique_sql_id	OUT bigint	Global unique ID.
enabled	OUT bool	Specifies whether the patch takes effect.
abort	OUT bool	Specifies whether the patch is an abort hint.
hint_str	OUT text	Hint text.

13.7.7 DBE_SQL_UTIL.create_hint_sql_patch

create_hint_sql_patch is an interface function used to create SQL patches for hints. It returns whether the execution is successful, as described in **Table 13-221**. This function is an overloaded function of the original function. The value of **parent_unique_sql_id** can be used to limit the effective range of the hint patch.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

Table 13-221 DBE_SQL_UTIL.create_hint_sql_patch input parameters and return values

Parameter	Type	Description
patch_name	IN name	Patch name.
unique_sql_id	IN bigint	Global unique ID.
parent_unique_sql_id	IN bigint	Global unique ID of the outer SQL statement. The value **0** indicates that the SQL patch statement outside the stored procedure is restricted to take effect. A non-zero value indicates that the specific stored procedure is restricted to take effect.
hint_string	IN text	Hint text.
description	IN text	Patch description. The default value is **NULL**.
enabled	IN bool	Specifies whether the patch takes effect. The default value is **true**.
result	OUT bool	Specifies whether this operation is successful.

13.7.8 DBE_SQL_UTIL.create_abort_sql_patch

create_abort_sql_patch is an interface function used to create abort SQL patches. It returns whether the execution is successful, as described in **Table 13-222**. This function is an overloaded function of the original function. The value of **parent_unique_sql_id** can be used to limit the effective range of the abort patch.

Only the initial user, sysadmin, opradmin, and monadmin have the permission to invoke this interface.

Table 13-222 DBE_SQL_UTIL.create_abort_sql_patch input parameters and return values

Parameter	Type	Description
patch_name	IN name	Patch name.
unique_sql_id	IN bigint	Global unique ID.
parent_unique_sql_id	IN bigint	Global unique ID of the outer SQL statement. The value **0** indicates that the SQL patch statement outside the stored procedure is restricted to take effect. A non-zero value indicates that the specific stored procedure is restricted to take effect.
description	IN text	Patch description. The default value is **NULL**.

Parameter	Type	Description
enabled	IN bool	Specifies whether the patch takes effect. The default value is **true**.
result	OUT bool	Specifies whether this operation is successful.

14 Configuring Running Parameters

14.1 Viewing Parameters

GaussDB uses a set of default GUC parameters after it is installed. You can modify the GUC parameters to enable GaussDB to better fit your service scenarios and data volume.

Procedure

Step 1 Connect to the database. For details, see "Database Quick Start > Connecting to a Database > Using gsql to Connect to a Database" in *Developer Guide*.

Step 2 View the GUC parameter values in the database.

- Method 1: Run the **SHOW** command.
  - View the value of a certain parameter.
    ```
    gaussdb=# SHOW server_version;
    ```
    server_version indicates the database version.
  - View values of all parameters.
    ```
    gaussdb=#  SHOW ALL;
    ```
- Method 2: Query the pg_settings view.
  - View the value of a certain parameter.
    ```
    gaussdb=# SELECT * FROM pg_settings WHERE NAME='server_version';
    ```
  - View values of all parameters.
    ```
    gaussdb=# SELECT * FROM pg_settings;
    ```

----End

Example

Check the character encoding type of the client.

```
gaussdb=# SHOW client_encoding;
 client_encoding
-----------------
 UTF8
(1 row)
```


14.2 Setting Parameters

Context

GaussDB provides multiple methods to set GUC parameters for databases, users, or sessions.

- Parameter names are case-insensitive.
- The parameter values can be integers, floating points, strings, Boolean values, or enumerated values.
  - The Boolean values can be **on**/**off**, **true**/**false**, **yes**/**no**, or **1**/**0**, and are case-insensitive.
  - The enumerated value range is specified in the **enumvals** column of the system catalog pg_settings.
- For parameters using units, specify their units during the setting. Otherwise, default units are used.
  - The default units are specified in the **unit** column of pg_settings.
  - The unit of memory can be KB, MB, or GB.
  - The unit of time can be ms, s, min, h, or d.

For details about parameters, see **GUC Parameters**.

Setting GUC Parameters

GaussDB provides six types of GUC parameters. For details about parameter types and their setting methods, see **Table 14-1**.

Table 14-1 GUC parameter types

Parameter Type	Description	Setting Method
INTERNAL	Fixed parameters. They are set during database creation and cannot be modified. Users can only view the parameters by running the **SHOW** command or in the pg_settings view.	None.
POSTMASTER	Database server parameters. They can be set when the database is started or in the configuration file.	Method 1 in **Table 14-2**.
SIGHUP	Global database parameters. They can be set when the database is started or be modified later.	Method 1 or 2 in **Table 14-2**.

Parameter Type	Description	Setting Method
BACKEND	Session connection parameters. They are specified during session connection creation and cannot be modified after that. The parameter setting becomes invalid when the session is disconnected. The parameters of this type are internal parameters and not recommended for users to set it.	Method 1 or 2 in **Table 14-2**. **NOTE** The parameter setting takes effect when the next session is created.
SUSET	Database administrator parameters. They can be set by common users during database startup or after the database is started. They can also be set by database administrators using SQL statements.	Method 1 or 2 by a common user, or method 3 by a database administrator in **Table 14-2**.
USERSET	Common user parameters. They can be set by any user at any time.	Method 1, 2, or 3 in **Table 14-2**.

You can set GUC parameters in GaussDB using the three methods listed in **Table 14-2**.

Table 14-2 Methods for setting GUC parameters

No.	Setting Method
Method 1	1. Modify a parameter. **gs_guc set -D** *datadir* **-c** "*paraname=value*" **NOTE** • If any parameter is a string variable, use **-c** *parameter*="'*value*'" or **-c** "*parameter* = '*value*'". • If any parameter is a string variable, gs_guc does not check the validity of the parameter. If the database runs abnormally due to invalid parameter settings, view gs_log to locate the fault. • Set a parameter for database nodes at the same time. **gs_guc set -Z datanode -N *all* -I *all* -c "*paraname=value*"** • Set a CM Agent parameter for database nodes. **gs_guc set -Z cmagent -c "paraname=value"** **gs_guc set -Z cmagent -N *all* -I *all* -c "paraname=value"** • Set a CM Server parameter for database nodes. **gs_guc set -Z cmserver -c "paraname=value"** **gs_guc set -Z cmserver -N *all* -I *all* -c "paraname=value"** 2. Restart the database to make the setting take effect. **NOTE** Restarting the database results in operation interruption. Properly plan the restart to avoid affecting users. **gs_om -t stop && gs_om -t start**

No.	Setting Method
Method 2	**gs_guc reload -D** *datadir* **-c** *"paraname=value"* **NOTE** Set a parameter for database nodes at the same time. **gs_guc reload -Z datanode -N *all* -I *all* -c "*paraname=value*"**
Method 3	Set parameters at the database, user, or session level. • Set a database-level parameter. gaussdb=# **ALTER DATABASE** *dbname* **SET** *paraname* **TO** *value;* The setting takes effect in the next session. • Set a user-level parameter. gaussdb=# **ALTER USER** *username* **SET** *paraname* **TO** *value;* The setting takes effect in the next session. • Set a session-level parameter. gaussdb=# **SET** *paraname* **TO** *value;* The parameter value in the current session is changed. After you exit the session, the setting becomes invalid. **NOTE** Session-level parameters set by SET have the highest priority, prior to those set by ALTER. Parameter values set by ALTER DATABASE have a higher priority than those set using ALTER USER. Priorities of the three methods are all higher than using gs_guc. The parameters set by ALTER DATABASE and ALTER USER do not take effect in the current session. You need to restart the session for the parameters to take effect.

CAUTION

- If you use method 1 or 2 to set a parameter that does not belong to the current node, the database displays a message indicating that the parameter is not supported.
- When you use method 3 to set a parameter, if the parameter value is an integer, leading zeros will be filtered out. For example, **SET** *paraname* **TO** *008192* and **SET** *paraname* **TO** *8192* have the same effect.

Procedure

The following example shows how to set **hot_standby** on the primary node of the database using method 1.

Step 1 Log in as the OS user **omm** to the primary node of the database.

Step 2 View the value of **hot_standby**.

```
cat /gaussdb/data/dbnode/gaussdb.conf | grep "hot_standby ="
hot_standby = on
```

The value **on** indicates that the query operation in the restoration phase is allowed.

Step 3 Set **hot_standby** to **off** to disable query operations in the restoration phase.

```
gs_guc set -Z datanode -D /gaussdb/data/dbnode -c "hot_standby=off"
```


NOTE

You can set **hot_standby** to **off** for the database nodes.

```
gs_guc set -Z datanode -N all -I all -c "hot_standby=off"
```

Step 4 Restart the database to make the setting take effect.

```
gs_om -t stop && gs_om -t start
```

Step 5 Connect to the database. For details, see "Database Quick Start > Connecting to a Database > Using gsql to Connect to a Database" in *Developer Guide*.

Step 6 Check whether the parameter is correctly set.

```
gaussdb=# SHOW hot_standby;
 hot_standby
-------------
 off
(1 row)
```

----End

The following example shows how to set **authentication_timeout** on a the primary node of the database using method 2:

Step 1 Log in as the OS user **omm** to the primary node of the database.

Step 2 View the value of **authentication_timeout**.

```
cat /gaussdb/data/dbnode/gaussdb.conf | grep authentication_timeout
authentication_timeout = 1min
```

Step 3 Set **authentication_timeout** to **59s**.

```
gs_guc reload -Z datanode -N all -I all -c "authentication_timeout = 59s"

Total instances: 2. Failed instances: 0.
Success to perform gs_guc!
```

NOTE

You can set **authentication_timeout** to **59s** for the database nodes.

```
gs_guc reload -Z datanode -N all -I all -c "authentication_timeout = 59s"
```

Step 4 Connect to the database. For details, see "Database Quick Start > Connecting to a Database > Using gsql to Connect to a Database" in *Developer Guide*.

Step 5 Check whether the parameter is correctly set.

```
gaussdb=# SHOW authentication_timeout;
 authentication_timeout
------------------------
 59s
(1 row)
```

----End

The following example shows how to set **explain_perf_mode** using method 3:

Step 1 Connect to the database. For details, see "Database Quick Start > Connecting to a Database > Using gsql to Connect to a Database" in *Developer Guide*.

Step 2 View the value of **explain_perf_mode**.

```
gaussdb=# SHOW explain_perf_mode;
 explain_perf_mode
-------------------
 normal
(1 row)
```


Step 3 Set **explain_perf_mode**.

Perform one of the following operations:

- Set a database-level parameter.
  ```
  gaussdb=# ALTER DATABASE postgres SET explain_perf_mode TO pretty;
  ```
  If the following information is displayed, the setting is successful:
  ```
  ALTER DATABASE
  ```
  The setting takes effect in the next session.
- Set a user-level parameter.
  ```
  gaussdb=# ALTER USER omm SET explain_perf_mode TO pretty;
  ```
  If the following information is displayed, the setting is successful:
  ```
  ALTER ROLE
  ```
  The setting takes effect in the next session.
- Set a session-level parameter.
  ```
  gaussdb=# SET explain_perf_mode TO pretty;
  ```
  If the following information is displayed, the setting is successful:
  ```
  SET
  ```

Step 4 Check whether the parameter is correctly set.

```
gaussdb=# SHOW explain_perf_mode;
 explain_perf_mode
---------------
 pretty
(1 row)
```

----End

Examples

- Example 1: Modify the maximum number of connections for the primary database node in GaussDB using method 1.
  a. Connect to the database. For details, see "Database Quick Start > Connecting to a Database > Using gsql to Connect to a Database" in *Developer Guide*.
  b. View the maximum number of connections.
     ```
     gaussdb=# SHOW max_connections;
      max_connections
     -----------------
      200
     (1 row)
     ```
  c. Exit the database.
     ```
     gaussdb=# \q
     ```
  d. Change the maximum number of connections for the primary database node in GaussDB.
     ```
     gs_guc set -Z datanode -N all -I all -c "max_connections = 800"
     ```
  e. Restart the database.
     ```
     gs_om -t stop && gs_om -t start
     ```
  f. Connect to the database. For details, see "Database Quick Start > Connecting to a Database > Using gsql to Connect to a Database" in *Developer Guide*.
  g. View the maximum number of connections.
     ```
     gaussdb=# SHOW max_connections;
      max_connections
     ```



```
-----------------
 800
(1 row)
```

- Example 2: Set **authentication_timeout** (timeout period for client authentication) for the primary node using method 2.

  a. Connect to the database. For details, see "Database Quick Start > Connecting to a Database > Using gsql to Connect to a Database" in *Developer Guide*.

  b. View the timeout period for client authentication.

  ```
  gaussdb=# SHOW authentication_timeout;
   authentication_timeout
  ------------------------
   1min
  (1 row)
  ```

  c. Exit the database.

  ```
  gaussdb=# \q
  ```

  d. Change the timeout period for client authentication of the primary node.

  ```
  gs_guc reload -Z datanode -N all -I all -c "authentication_timeout = 59s"
  ```

  e. Connect to the database. For details, see "Database Quick Start > Connecting to a Database > Using gsql to Connect to a Database" in *Developer Guide*.

  f. View the timeout period for client authentication.

  ```
  gaussdb=# SHOW authentication_timeout;
   authentication_timeout
  ------------------------
   59s
  (1 row)
  ```

- Example 3: Change the maximum number of connections between GaussDB database nodes.

  a. Connect to the database. For details, see "Database Quick Start > Connecting to a Database > Using gsql to Connect to a Database" in *Developer Guide*.

  b. View the maximum number of connections.

  ```
  gaussdb=# SHOW max_connections;
   max_connections
  -----------------
   200
  (1 row)
  ```

  c. Exit the database.

  ```
  gaussdb=# \q
  ```

  d. Change the maximum number of connections between GaussDB database nodes.

  ```
  gs_guc set -Z datanode -N all -I all -c "max_connections = 500"
  ```

  e. Restart the database.

  ```
  gs_om -t stop
  gs_om -t start
  ```

  f. Connect to the database. For details, see "Database Quick Start > Connecting to a Database > Using gsql to Connect to a Database" in *Developer Guide*.

  g. View the maximum number of connections.

  ```
  gaussdb=# SHOW max_connections;
   max_connections
  -----------------
   500
  (1 row)
  ```


- Example 4: Set **authentication_timeout** (timeout period for client authentication) for database nodes.
  a. Connect to the database. For details, see "Database Quick Start > Connecting to a Database > Using gsql to Connect to a Database" in *Developer Guide*.
  b. View the timeout period for client authentication.
  ```
  gaussdb=# SHOW authentication_timeout;
   authentication_timeout
  ------------------------
   1min
  (1 row)
  ```
  c. Exit the database.
  ```
  gaussdb=# \q
  ```
  d. Change the timeout period for client authentication of GaussDB database nodes.
  ```
  gs_guc reload -Z datanode -N all -I all -c "authentication_timeout = 30s"
  ```
  e. Connect to the database. For details, see "Database Quick Start > Connecting to a Database > Using gsql to Connect to a Database" in *Developer Guide*.
  f. View the timeout period for client authentication.
  ```
  gaussdb=# SHOW authentication_timeout;
   authentication_timeout
  ------------------------
   30s
  (1 row)
  ```

14.3 GUC Parameters

14.3.1 GUC Parameter Usage

A database provides many GUC parameters. Changing these parameters will affect the behavior of the database system. Before modifying these parameters, learn the impact of these parameters on the database. Otherwise, unexpected results may occur.

Precautions

- If the value range of a parameter is a string, the string should comply with the path and file naming conventions of the OS running the target database.
- If the maximum value of a parameter is *INT_MAX*, the maximum parameter value varies by OS.
- If the maximum value of a parameter is *DBL_MAX*, the maximum parameter value varies by OS.
- Some GUC parameters affect the selection of functions and operators, compilation of stored procedures, and generation of execution plans. As a result, views, default values of function parameters, compilation products of stored procedures, and plan cache are affected. Due to these mechanisms, subsequent GUC parameter changes may not affect these behaviors.
  – View: When defining a view, the database generates rewriting rules based on the GUC parameter status. Subsequent operations on the view directly follow the corresponding rules without being affected by GUC parameters (which affect the generation of rewriting rules).

- Default value of a function parameter: For a function parameter, a function is selected as the default parameter value based on the GUC parameter status when the function parameter is created and the OID of the function that is used as the default value is recorded in the corresponding system catalog (pg_proc). When this function parameter is executed later, the function OID recorded in the system catalog will be used as the default value, and the value is not affected by GUC parameters.
- Compilation product of a stored procedure: When compiling a stored procedure, the database generates a compilation product based on the GUC parameter status. When executing the stored procedure, the database directly uses the compilation product without being affected by GUC parameters (which affect the generation of compilation products).
- Plan cache: When executing an SQL statement, the database generates an execution plan based on the GUC parameter status. If the plan is cached, the SQL statement will be executed according to the plan without being affected by GUC parameters (which affect the generation of execution plans).

The following table lists the GUC parameters that exert influence. You can view the function of each GUC parameter.

GUC Parameter	Enabled	Disabled	Impact	Example
convert_string_to_digit	Strings are preferentially converted to numbers.	Character strings cannot be preferentially converted to numbers.	Views, default values of function parameters, stored procedure compilation products, and plan cache.	CREATE or REPLACE FUNCTION test(c numeric) RETURN text package AS BEGIN RETURN 'test(c numeric)'; END; / CREATE or REPLACE FUNCTION test(c varchar2) RETURN text package AS BEGIN RETURN 'test(c varchar2)'; END; / SET convert_string_to_digit=on; CREATE or REPLACE VIEW test_view AS SELECT test('123'::text); SELECT test('123'::text); test ----------------- test(c numeric) (1 row) SELECT * FROM test_view; test ----------------- test(c numeric) (1 row) -- Disable the parameter. SET convert_string_to_digit=off; SELECT test('123'::text); test ------------------ test(c varchar2) (1 row) -- The view behavior is inconsistent with the direct function calling behavior. SELECT * FROM test_view; test ----------------- test(c numeric) (1 row))

GUC Parameter	Enabled	Disabled	Impact	Example
set behavior_compat_options = 'current_sysdate';	The SYSDATE function obtains the current OS time. (The current_sysdate function is used at the bottom layer.)	The SYSDATE function obtains the current database time. (The sysdate function is used at the bottom layer.)	Views, default values of function parameters, stored procedure compilation products, and plan cache.	SET behavior_compat_options = 'current_sysdate'; CREATE or REPLACE VIEW test_view_SYSDATE AS SELECT SYSDATE AS test; SELECT * FROM test_view_SYSDATE;--Returns the current OS time. SELECT SYSDATE;--Returns the current OS time. -- Disable the parameter. SET behavior_compat_options = ''; -- The view behavior is inconsistent with the direct function calling behavior. SELECT * FROM test_view_SYSDATE;--Returns the current OS time. SELECT SYSDATE;--Returns the current database time.

GUC Parameter	Enabled	Disabled	Impact	Example
set a_format_version='10c'; set a_format_dev_version='s1';	• NVL2 is supported. • Some system functions are added. For details, see the disabled system functions supported by the **a_format_disable_func** parameter. • The CASE WHEN return type is changed. • Constructors of the set type are prior to functions. • Implicit conversion from timestamp to timestamptz is supported.	The functions supported when the parameter is enabled are not supported.	Views, default values of function parameters, stored procedure compilation products, and plan cache.	SET a_format_version='10c'; SET a_format_dev_version='s1'; CREATE OR REPLACE VIEW test_view_nvl2 AS SELECT nvl2(1,2,3) AS test; SELECT * FROM test_view_nvl2; test ------ 2 (1 row) RESET a_format_dev_version; SELECT * FROM test_view_nvl2; test ------ 2 (1 row)

GUC Parameter	Enabled	Disabled	Impact	Example
set a_format_version='10c'; set a_format_dev_version='s2';	• CURRENT_TIMESTAMP '(' FCONST ')' is supported. • DBTIMEZONE is supported. • LNNVL is supported. • CURRENT_DATE (when **a_format_date_timestamp** is set to **off**): returns the date and time (timestamp). • Some system functions are added. For details, see the disabled system functions supported by the **a_format_disabl**	The functions supported when the parameter is enabled are not supported.	Views, default values of function parameters, stored procedure compilation products, and plan cache.	SET a_format_version='10c'; SET a_format_dev_version='s2'; CREATE or REPLACE VIEW test_view_LNNVL AS SELECT LNNVL(123=123) AS test; SELECT * FROM test_view_LNNVL; RESET a_format_dev_version; SELECT * FROM test_view_LNNVL;

GUC Parameter	Enabled	Disabled	Impact	Example
	e_func parameter.			
set a_format_version='10c'; set a_format_dev_version='s4';	Some system functions are added. For details, see the disabled system functions supported by the **a_format_disable_func** parameter.	The functions supported when the parameter is enabled are not supported.	Views, default values of function parameters, stored procedure compilation products, and plan cache.	-

GUC Parameter	Enabled	Disabled	Impact	Example
set a_format_version='10c'; set a_format_dev_version='s5';	• Some system functions are added. For details, see the disabled system functions supported by the **a_format_disable_func** parameter. • Composite constructors are prior to functions.	The functions supported when the parameter is enabled are not supported.	Views, default values of function parameters, stored procedure compilation products, and plan cache.	--The composite type constructor is prior to functions. CREATE TYPE tt AS (val1 int, val2 int); CREATE OR REPLACE FUNCTION tt(va int, vb int) RETURN int IS ret int; BEGIN ret := va; RETURN ret; END; / SET a_format_version='10c'; SET a_format_dev_version='s5'; -- Assign a value to the *tt* variable. CREATE OR REPLACE FUNCTION test0 RETURN int IS va tt; ret int; BEGIN va := tt(1,2); RAISE INFO 'tt: %' ,va; ret := 9; RETURN ret; END; / SELECT test0(); INFO: tt: (1,2) test0 ------- 9 (1 row) SET a_format_version=''; SET a_format_dev_version=''; SELECT test0(); INFO: tt: (1,2) test0 ------- 9 (1 row) -- Assign a value to the *tt* variable. CREATE OR REPLACE FUNCTION test0 RETURN int IS va tt; ret int; BEGIN va := tt(1,2); RAISE INFO 'tt: %' ,va; ret := 9; RETURN ret; END; / select test0();

GUC Parameter	Enabled	Disabled	Impact	Example
				ERROR: cannot assign non-composite value to a row variable

GUC Parameter	Enabled	Disabled	Impact	Example
set behavior_compat_options = 'enable_bpcharlikebpchar_compare';	The bpcharlikebpchar and bpcharnlikebpchar operators are enabled.	The bpcharlikebpchar and bpcharnlikebpchar operators are disabled.	Views, stored procedure compilation products, and plan cache.	CREATE TABLE op_test (col BPCHAR(2) DEFAULT NULL); CREATE INDEX op_index ON op_test(col); INSERT INTO op_test VALUES ('a'); INSERT INTO op_test VALUES ('1'); INSERT INTO op_test VALUES ('11'); INSERT INTO op_test VALUES ('12'); INSERT INTO op_test VALUES ('sd'); INSERT INTO op_test VALUES ('aa'); SET behavior_compat_options = 'enable_bpcharlikebpchar_compare'; EXPLAIN (COSTS OFF) SELECT * FROM op_test WHERE col LIKE col::BPCHAR ORDER BY col; QUERY PLAN ------------------------------ Sort Sort Key: col -> Seq Scan on op_test Filter: (col ~~ col) (4 rows) CREATE OR REPLACE VIEW test_view_bpchar AS SELECT * FROM op_test WHERE col LIKE col::BPCHAR ORDER BY col; EXPLAIN SELECT * FROM test_view_bpchar; QUERY PLAN -- Sort (cost=34.48..34.50 rows=10 width=12) Sort Key: op_test.col -> Seq Scan on op_test (cost=0.00..34.31 rows=10 width=12) Filter: (col ~~ col) (4 rows) SET behavior_compat_options = ''; EXPLAIN (COSTS OFF) SELECT * FROM op_test WHERE col LIKE col::BPCHAR ORDER BY col; QUERY PLAN -------------------------------------- Sort Sort Key: col -> Seq Scan on op_test Filter: (col ~~ (col)::text) (4 rows) EXPLAIN SELECT * FROM test_view_bpchar; QUERY PLAN --

GUC Parameter	Enabled	Disabled	Impact	Example
				Sort (cost=34.48..34.50 rows=10 width=12) Sort Key: op_test.col -> Seq Scan on op_test (cost=0.00..34.31 rows=10 width=12) Filter: (col OPERATOR(pg_catalog.~~) col) (4 rows) -- Rebuild a view when the parameter is disabled. CREATE OR REPLACE VIEW test_view_bpchar AS SELECT * FROM op_test WHERE col LIKE col::BPCHAR ORDER BY col; explain select * from test_view_bpchar; QUERY PLAN ---------- Sort (cost=39.34..39.37 rows=10 width=12) Sort Key: op_test.col -> Seq Scan on op_test (cost=0.00..39.17 rows=10 width=12) Filter: (col ~~ (col)::text) (4 rows)
set b_format_version='5.7'; set b_format_dev_version='s1';	• CURDATE, CURRENT_TIME, CURRENT_TIMESTAMP, LOCALTIME, and LOCALTIMESTAMP NOW return the statement execution time. • SYSDATE returns the function execution time.	The functions return the transaction start time.	Views, default values of function parameters, stored procedure compilation products, and plan cache.	-

GUC Parameter	Enabled	Disabled	Impact	Example
set b_format_version='5.7'; set b_format_dev_version='s1';	The behaviors of the DB_JSOBJ, LAST_DAY_FUNC, EXTRACT, TIMESTAMPDIFF, and SUBSTRING functions are changed to be the same as that of a B-compatible database.	Restores the default behavior.	Views, default values of function parameters, stored procedure compilation products, and plan cache.	-

GUC Parameter	Enabled	Disabled	Impact	Example
set behavior_compat_options = 'bind_procedure_searchpath';	• Specifies the search path of the database objects in a stored procedure for which no schema name is specified. • If no schema name is specified for a stored procedure, the schema to which the stored procedure belongs is searched preferentially.	Restores the default behavior.	Stored procedure compilation products.	-

GUC Parameter	Enabled	Disabled	Impact	Example
set behavior_compat_options = 'enable_out_param_override';	Determines the overloading of output parameters of a stored procedure.	Restores the default behavior.	Stored procedure compilation products.	-
enable_bitmapscan	The bitmapscan operator is selected.	The bitmapscan operator is not selected.	Plan cache.	SET enable_auto_explain=true; SET auto_explain_level=notice; CREATE TABLE t1(a int, b int, c int,d varchar)WITH(storage_type=ustore); CREATE TABLE t2(a int, b int, c int,d varchar)WITH(storage_type=ustore); INSERT INTO t1 SELECT i,10*random()*i,i%20,'atr'||i from generate_series(1,1000) i; CREATE INDEX index1 ON t1(a); SET enable_bitmapscan=on; SET enable_seqscan=off; SET enable_indexscan=off; CREATE OR REPLACE PROCEDURE proc_while_loop() AS DECLARE i int :=1; BEGIN WHILE i < 5 LOOP INSERT INTO t2 SELECT * FROM t1 WHERE a = i ; i:=i+1; END LOOP; END; / CALL proc_while_loop(); SET enable_bitmapscan=off; CALL proc_while_loop(); SET enable_bitmapscan=on; CALL proc_while_loop();

GUC Parameter	Enabled	Disabled	Impact	Example
enable_indexscan	The indexscan operator is selected.	The indexscan operator is not selected.	Plan cache.	SET enable_auto_explain=true; SET auto_explain_level=notice; CREATE TABLE t1(a int, b int, c int,d varchar)WITH(storage_type=ustore); CREATE TABLE t2(a int, b int, c int,d varchar)WITH(storage_type=ustore); INSERT INTO t1 SELECT i,10*random()*i,i%20,'atr'||i from generate_series(1,1000) i; CREATE INDEX index1 on t1(a); SET enable_bitmapscan=off; SET enable_seqscan=off; CREATE OR REPLACE PROCEDURE proc_while_loop() AS DECLARE i int :=1; BEGIN WHILE i < 5 LOOP insert into t2 select * from t1 where a = i ; i:=i+1; END LOOP; END; / CALL proc_while_loop(); SET enable_indexscan=off; CALL proc_while_loop();

GUC Parameter	Enabled	Disabled	Impact	Example
query_Dop	Users can specify the degree of query parallelism. If the SMP function is enabled, the system uses the specified degree of parallelism.	The default value is **1**, which indicates that parallel query is disabled.	Plan cache.	SET enable_auto_explain=true; SET auto_explain_level=notice; CREATE TABLE t1(a int, b int, c int,d varchar)WITH(storage_type=ustore); CREATE TABLE t2(a int, b int, c int,d varchar)WITH(storage_type=ustore); INSERT INTO t1 SELECT i,10*random()*i,i%20,'atr'||i from generate_series(1,1000) i; CREATE INDEX index1 ON t1(a); SET enable_bitmapscan=f; CREATE OR REPLACE PROCEDURE proc_while_loop() AS DECLARE i int :=1; BEGIN WHILE i < 5 LOOP INSERT INTO t2 SELECT /*+ indexscan(t1 index1) */* FROM t1 WHERE a = i ; i:=i+1; END LOOP; END; / SET enable_auto_explain=true; SET auto_explain_level=notice; SET client_min_messages=log; SET query_dop=4; SET sql_beta_feature='enable_plsql_smp'; CALL proc_while_loop(); SET enable_force_Smp=on; CALL proc_while_loop(); SET query_Dop=6; CALL proc_while_loop();

GUC Parameter	Enabled	Disabled	Impact	Example
set plan_cache_mode = force_generic_plan	In PBE mode, the gplan is forcibly executed.	In PBE mode, the gplan is not forcibly executed.	Plan cache.	SET enable_auto_explain=true; SET auto_explain_level=notice; CREATE TABLE t1(a int, b int, c int,d varchar)WITH(storage_type=ustore); CREATE TABLE t2(a int, b int, c int,d varchar)WITH(storage_type=ustore); INSERT INTO t1 SELECT i,10*random()*i,i%20,'atr'||i from generate_series(1,1000) i; CREATE INDEX index1 ON t1(a); SET enable_bitmapscan=on; SET enable_seqscan=off; SET enable_indexscan=off; CREATE OR REPLACE PROCEDURE proc_while_loop() AS DECLARE i int :=1; BEGIN WHILE i < 5 LOOP insert into t2 select * from t1 where a = i ; i:=i+1; END LOOP; END; / CALL proc_while_loop(); SET plan_cache_mode = force_generic_plan; PREPARE aa AS SELECT proc_while_loop(); EXECUTE aa; SET enable_bitmapscan=off; EXECUTE aa;
a_format_date_timestamp	CURRENT_DATE returns the timestamp when the current SQL statement is started.	CURRENT_DATE returns the date or date and time when the transaction is started.	Views, default values of function parameters, stored procedure compilation products, and plan cache.	SET a_format_date_timestamp=on; CREATE OR REPLACE PROCEDURE proc_while_loop() AS DECLARE i date; BEGIN i:=current_date; raise info 'step2:%',TO_CHAR(current_date, 'MM-DD-YYYY HH24:MI:SS'); END; / CALL proc_while_loop(); SET a_format_date_timestamp=off; CALL proc_while_loop();

GUC Parameter	Enabled	Disabled	Impact	Example
proc_implicit_for_loop_variable	Determines the behavior of the FOR_LOOP query statement in a stored procedure. When this parameter is set, if *rec* has been defined in the FOR rec IN query LOOP statement, the defined *rec* variable is not reused but a new variable is created. Otherwise, the defined *rec* variable is reused and no variable is created.	Restores the default behavior.	Stored procedure compilation products.	-
adjust_systemview_priority	Views with the same name preferentially access user-defined views.	Views with the same name preferentially access system views.	Default value of parameters in the views.	CREATE DATABASE database_test DBCOMPATIBILITY = 'A'; SELECT datname,datcompatibility,dattimezone FROM pg_database; \c database_test CREATE TABLE t1(c int); CREATE VIEW dual AS SELECT * FROM t1; SHOW adjust_systemview_priority; SELECT * FROM dual; SET adjust_systemview_priority = on; SELECT * FROM dual;

14.3.2 File Location

After a database has been installed, three configuration files (**gaussdb.conf**, **gs_hba.conf**, and **gs_ident.conf**) are automatically generated and saved in the data directory. You can use the methods described in this section to change the names and save paths of these configuration files.

When changing the storage directory of a configuration file, set **data_directory** in **gaussdb.conf** to the actual data directory.

NOTICE

If a configuration file is incorrectly modified, the database will be seriously affected. Do not modify the configuration files mentioned in this section after installation.

data_directory

Parameter description: Specifies the GaussDB **data** directory. Only the sysadmin user can access this parameter. You can set this parameter using one of the following methods:

- Set it when you install the GaussDB.
- This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Value range: a string, consisting of one or more characters.

Default value: Specify this parameter during installation. If this parameter is not specified during installation, the database is not initialized by default.

config_file

Parameter description: Specifies the configuration file (**gaussdb.conf**) of the primary node.

This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Value range: a string, consisting of one or more characters.

Default value: **gaussdb.conf** (The absolute directory of this file may be displayed in the actual situation.)

hba_file

Parameter description: Specifies the configuration file (**gs_hba.conf**) for host-based authentication (HBA). This parameter can be specified only in the **gaussdb.conf** file and can be accessed only by the sysadmin user.

This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Value range: a string.

Default value: **gs_hba.conf** (The absolute directory of this file may be displayed in the actual situation.)

ident_file

Parameter description: Specifies the name of the configuration file (**gs_ident.conf)** for client authentication. Only the sysadmin user can access this parameter.

This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Value range: a string.

Default value: **gs_ident.conf** (The absolute directory of this file may be displayed in the actual situation.)

external_pid_file

Parameter description: Specifies the extra PID file that can be used by the server management program. Only the sysadmin user can access this parameter.

This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

NOTICE

This parameter takes effect only after the database restarts.

Value range: a string.

Default value: empty

14.3.3 Connection and Authentication

14.3.3.1 Connection Settings

This section describes parameters related to client-server connection modes.

light_comm

Parameter description: Specifies whether the server uses the lightweight communications mode.

This parameter specifies whether the server uses the communication mode based on lightweight locks and non-blocking sockets.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: The lightweight communications mode is used.
- **off**: The lightweight communications mode is not used.

Default value: **off**

Setting method: This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Setting suggestion: none

Risks and impacts of improper settings: none

listen_addresses

Parameter description: Specifies the TCP/IP address of the client for a server to listen on.

This parameter specifies the IP address used by the GaussDB server for listening, for example, an IPv4 address. Multiple NICs may exist on the host and each NIC can be bound to multiple IP addresses. This parameter specifies the IP addresses to which GaussDB is bound. The client can use the IP addresses specified by this parameter to connect to GaussDB or send requests to GaussDB.

Parameter type: string.

Unit: none

Value range:

- Host name or IP address. Multiple values are separated with commas (,).
- Asterisk (*) or **0.0.0.0**, indicating that all IP addresses will be listened on, which is not recommended due to potential security risks. This parameter must be used together with valid addresses (for example, the local IP address). Otherwise, the build may fail. In primary/standby mode, if the value is set to ***** or **0.0.0.0**, the value of **localport** in the **gaussdb.conf** file of the database on the primary node cannot be the value of **dataPortBase** plus 1. Otherwise, the database cannot be started.
- If the parameter value is invalid (for example, the parameter value contains invalid IP addresses or characters, is empty, or is listened repeatedly), the process fails to be started.

Default value: After the database instance is installed, the default value is set based on the IP addresses of different instances in the **public_cloud.conf** configuration file. The default value of DN is "*IP address of the data.net NIC*".

Setting method: This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

Setting suggestion: Retain the default value.

NOTE

- The **public_cloud.conf** file contains the following NIC information: **mgr.net** (management NIC), **data.net** (data NIC), and **virtual.net** (virtual NIC).
- If an IPv6 address is used and the IP address is a fe80 address block, add '%zone index' during the configuration.
- This parameter can be set in **gs_guc reload** mode. The GaussDB kernel dynamically listens on the IP addresses specified by this parameter based on the policy of listening on new IP addresses, keeping listening on duplicate IP addresses, and disabling listening on invalid IP addresses. If an invalid IP address exists (for example, the IP address is not configured on the local host, the IP address is invalid, or the IP address has been listened on), the kernel fails to listen on the IP address. In this case, the value of **listen_addresses** does not match the actual listening IP address.
- If this parameter is set in **gs_guc reload** mode and all IP addresses in the parameter value are invalid, listening on new IP addresses fails and listening on all old IP addresses is canceled. If this parameter is left empty, new IP addresses are intercepted and old IP addresses are listened on.
- If **listen_addresses** is configured in the process startup parameter, this parameter is forcibly set to the configured value and will not be modified by reload.
- If the new parameter value is the same as the old parameter value, the listening action is not performed. Therefore, when the **listen_addresses** parameter is set and an invalid IP address exists in the list, the kernel fails to listen on the IP address. After the IP address becomes valid and the same parameter is set again, the kernel does not listen on the IP address because the two parameter values are the same. In this case, you need to run the **gs_guc reload** command to remove the IP address from the parameter value and add the IP address again.
- After this parameter is updated by using **gs_guc reload**, if an IP address is no longer listened on, you need to execute the gs_validate_ext_listen_ip() function to clear the connection. For details about the input parameters and execution, see "SQL Reference > Functions and Operators > Other System Functions" in *Developer Guide*.
- Do not use gs_guc to reload the value of **listen_addresses** on all DNs in **"-N all"** mode. This parameter can be set only on a single DN.
- Dynamically modifying **listen_addresses** is a high-risk operation. If the configuration is incorrect, the database may fail to accept new connections, affecting services. Exercise caution when performing this operation.

Risks and impacts of improper settings: none

local_bind_address

Parameter description: Specifies the local IP address bound to the current node for connecting to other nodes in the database.

Parameter type: string.

Unit: none

Value range: an IPv4 address. Multiple IP addresses are not supported.

Default value: After the database instance is installed, the default value is set based on the IP addresses of different instances in the **public_cloud.conf** configuration file. The default value of DN is "*IP address of the data.net NIC*".

NOTE

The **public_cloud.conf** file contains the following NIC information: **mgr.net** (management NIC), **data.net** (data NIC), and **virtual.net** (virtual NIC).

Setting method: This is a POSTMASTER parameter. Set it based on instructions in Table 14-1.

Setting suggestion: Retain the default value.

Risks and impacts of improper settings: none

port

Parameter description: Specifies the TCP port listened on by the GaussDB.

Parameter type: integer.

Unit: none

Value range: 1 to 65535

NOTE

- When setting the port number, ensure that the port number is not in use. When setting the port numbers of multiple instances, ensure that the port numbers do not conflict.
- Ports 1 to 1023 are reserved for the OS. Do not use them.
- When the database instance is installed using the configuration file, pay attention to the ports reserved in the communication matrix in the configuration file. For example, the port specified by the value of **dataPortBase** plus 1 needs to be reserved for internal tools Therefore, during database instance installation, the maximum port number is **65529** for DNs. Ensure that the port number does not conflict with each other.

Default value: **5432** (The actual value is specified in the configuration file during installation.)

Setting method: This is a POSTMASTER parameter. Set it based on instructions in Table 14-1.

Setting suggestion: Retain the default value.

Risks and impact of improper settings: This parameter is specified in the configuration file during installation. Do not modify this parameter unless necessary. Otherwise, database communications will be affected after the modification.

max_connections

Parameter description: Specifies the maximum number of concurrent connections to the database. This parameter influences the concurrent processing capability of the database.

Parameter type: integer.

Unit: none

Value range: an integer. The minimum value is **10** (greater than the value of **max_wal_senders**). The theoretical maximum value is **262143**. The actual maximum value is a dynamic value, which is calculated using the formula: 262143 – **job_queue_processes** – **autovacuum_max_workers** – **AUXILIARY_BACKENDS** – **AV_LAUNCHER_PROCS** – **max_inner_tool_connections** – **max_concurrent_autonomous_transactions** – *min(max(newValue/4,64),1024)*. The values of **job_queue_processes**, **autovacuum_max_workers**,

max_inner_tool_connections, and **max_concurrent_autonomous_transactions** depend on the settings of the corresponding GUC parameters. **AUXILIARY_BACKENDS** indicates the number of reserved auxiliary threads, which is fixed to **20**. **AV_LAUNCHER_PROCS** indicates the number of reserved autovacuum launcher threads, which is fixed to **2**. In *min(max(newValue/4,64),1024)*, **newValue** indicates the new value.

The value range of this parameter varies depending on different instance memory specifications.

Table 14-3 Value ranges of memory specifications for different instances

Memory Specifications	Value Range for DNs
< 32 GB	[10, 300]
[32GB,64GB)	[10, 600]
[64GB,128GB)	[10,2048]
[128GB,256GB)	[10,5000]
[256GB,480GB)	[10,11000]
[480GB,512GB)	[10,24000]
[512GB,640GB)	[10,25000]
[640GB,768GB)	[10,34000]
[768GB,1024GB)	[10,40000]
[1024GB,1536GB)	[10,55000]
[1536GB,2048GB)	[10,85000]
≥ 2048 GB	[10,110000]

Default value:

85000 (196-core CPU/1536 GB memory); **55000** (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory); **40000** (96-core CPU/768 GB memory); **34000** (80-core CPU/640 GB memory); **25000** (64-core CPU/512 GB memory); **24000** (60-core CPU/480 GB memory); **11000** (32-core CPU/256 GB memory); **5000** (16-core CPU/128 GB memory); **2048** (8-core CPU/64 GB memory); **600** (4-core CPU/32 GB memory); **300** (4-core CPU/16 GB memory)

Setting method: This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Setting suggestion: Retain the default value of this parameter in the primary node of the databases. If this parameter is set to a smaller value, the number of available connections decreases. Before changing the value of this parameter, understand the parameter description and exercise caution to avoid risks caused by misoperations.

Risks and impacts of improper settings:

- If the value of **max_connections** exceeds the maximum dynamic value, the node fails to be started and the following error message is displayed: "invalid value for parameter "max_connections"." Alternatively, the memory fails to be allocated during the node startup and the following error message is displayed: "Cannot allocate memory."
- If the value of **max_connections** is not increased based on the external export specifications, the memory parameter is not adjusted in the same proportion. When the service load is heavy, the memory may be insufficient and the error message "memory is temporarily unavailable" is displayed.

NOTE

- If the number of connections of the administrator exceeds the value of **max_connections**, the administrator can still connect to the database after the connections are used up by common users. If the number of connections exceeds the value of **sysadmin_reserved_connections**, an error is reported. That is, the maximum number of connections of the administrator is equal to the value of **max_connections** + **sysadmin_reserved_connections**.
- For common users, internal jobs use some connections. Therefore, the value of this parameter is slightly less than that of **max_connections**. The value depends on the number of internal connections.

max_inner_tool_connections

Parameter description: Specifies the maximum number of concurrent connections of a tool which is allowed to connect to the database. This parameter influences the concurrent connection capability of the GaussDB tool.

Parameter type: integer.

Unit: none

Value range: The minimum value is **1**. The formula for calculating the maximum value: 262143 – **job_queue_processes** – **autovacuum_max_workers** – **max_connections** – **max_concurrent_autonomous_transactions** – **AUXILIARY_BACKENDS** – **AV_LAUNCHER_PROCS** – *min(max(max_connections/4,64),1024)*. The values of **job_queue_processes**, **autovacuum_max_workers**, **max_connections**, and **max_concurrent_autonomous_transactions** are related to the settings of the corresponding GUC parameters. **AUXILIARY_BACKENDS** indicates the number of reserved auxiliary threads and its value is fixed to **20**. **AV_LAUNCHER_PROCS** indicates the number of launcher threads reserved for autovacuum and its value is fixed to **2**.

Default value: **50** (196-core CPU/1536 GB memory, 128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 64-core CPU/512 GB memory, 60-core CPU/480 GB memory, 32-core CPU/256 GB memory, 16-core CPU/128 GB memory, 8-core CPU/64 GB memory, 4-core CPU/32 GB memory); **10** (4-core CPU/16 GB memory)

If the default value is greater than the maximum value supported by the kernel (determined when the **gs_initdb** command is executed), an error message is displayed.

Setting method: This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Setting suggestion: Retain the default value of this parameter in the primary database node file. If this parameter is set to a large value, GaussDB requires more System V shared memories or semaphores, which may exceed the default maximum configuration of the OS. In this case, modify the value as needed.

Risks and impacts of improper settings: none

sysadmin_reserved_connections

Parameter description: Specifies the minimum number of connections reserved for administrators. You are advised not to set this parameter to a large value. This parameter is used together with the **max_connections** parameter. The maximum number of connections of the administrator is equal to the value of **max_connections** + **sysadmin_reserved_connections**.

NOTE

- When the thread pool function is enabled, if the thread pool is fully occupied, a processing bottleneck occurs. As a result, connections reserved by the administrator cannot be established. In this case, you can use gsql to establish connections through the port specified by the primary port number plus 1 and clear useless sessions.
- When the reserved connections are used up, new connections will fail. In this case, you can only restart the database to restore the connections. Therefore, exercise caution when using the reserved connections.

Parameter type: integer.

Unit: none

Value range: an integer ranging from 0 to *MIN(262143, max_connections)* (the smaller value between **262143** and **max_connections**). For details about how to calculate the value of **max_connections**, see the preceding description.

Default value: **3**

Setting method: This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Setting suggestion: none

Risks and impacts of improper settings: none

service_reserved_connections

Parameter description: Specifies the minimum number of connections reserved for background O&M users (with the persistence attribute). A large value is not recommended. This parameter is used together with **max_connections**. The maximum number of connections of an O&M user can be calculated as follows: **max_connections** + **service_reserved_connections**.

Parameter type: integer.

Unit: none

Value range: 0 to 262143.

Default value: **10**

Setting method: This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Setting suggestion: Retain the default value. If this parameter is set to a small value, the O&M user (with the persistence attribute) cannot connect to the database and jobs cannot be executed if **max_connections** is set to the maximum value.

Risks and impacts of improper settings: none

unix_socket_directory

Parameter description: Specifies the UDS directory for the GaussDB server to listen to connections from the client.

Parameter type: string.

Unit: none

Value range: valid path of a directory.

NOTE

The value of this parameter depends on the maximum path length of a directory in the OS. If the value exceeds the limit, "Unix-domain socket path "xxx" is too long" is reported.

Default value: empty. The actual value is specified by the configuration file during installation.

Setting method: This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Setting suggestion: none

Risks and impact of improper configuration: If the configuration is incorrect (for example, the length exceeds the limit or the directory is invalid), the process cannot be started properly. You can locate the fault based on the startup logs.

unix_socket_group

Parameter description: Specifies the group of the UDS (the user of a socket is the user who starts the server). This parameter can work with **unix_socket_permissions** to control socket access.

This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Value range: a string. If this parameter is set to an empty string, the default group of the current user is used.

Default value: an empty string.

unix_socket_permissions

Parameter description: Specifies access permissions on the UDS.

The UDS uses the usual permission set of the Unix file system. The value of this parameter should be a number (acceptable for the **chmod** and **umask** commands). If a user-defined octal format is used, the number must start with 0.

You are advised to set it to **0770** (only allowing access from users connecting to the database and users in the same group as them) or **0700** (only allowing access from users connecting to the database).

This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Value range: 0000 to 0777

Default value: **0700**

NOTE

In the Linux OS, a document has one document attribute and nine permission attributes, which are the read (r), write (w), and execute (x) permissions of the **owner**, **group**, and **others** groups.

The r, w, and x permissions are represented by the following numbers:

r: 4

w: 2

x: 1

-: 0

The three attributes in a group are accumulative.

For example, **-rwxrwx---** indicates the following permissions:

owner=rwx=4+2+1=7

group=rwx=4+2+1=7

others =---=0+0+0=0

The permission of the file is 0770.

application_name

Parameter description: Specifies the client name used in the current connection request.

This is a USERSET parameter. Set it based on instructions in **Table 14-1**. Note that after the client is connected, this parameter is set to the client name and the client level. Therefore, only the session-level parameters take effect. Other methods do not take effect because they are lower than the client level.

When a standby node requests to replicate logs on the primary node, if this parameter is not an empty string, it is used as the name of the streaming replication slot of the standby node on the primary node. In this case, if the length of this parameter exceeds 61 bytes, only the first 61 bytes are used as the streaming replication slot name.

Value range: a string.

Default value: empty (The actual value is the name of the application connected to the backend.)

connection_info

Parameter description: Specifies the database connection information, including the driver type, driver version, driver deployment path, and process owner.

This is a USERSET parameter used for O&M. You are advised not to set the parameter.

Value range: a string.

Default value: an empty string.

NOTE

- An empty string indicates that the driver connected to the database does not support automatic setting of the **connection_info** parameter or the parameter is not set by users in applications.
- The following is an example of the concatenated value of **connection_info**:
  ```
  {"driver_name":"ODBC","driver_version": "(GaussDB Kernel XXX.X.XXX build 290d125f) compiled at 2020-05-08 02:59:43 commit 2143 last mr 131 release","driver_path":"/usr/local/lib/gsqlodbcw.so","os_user":"omm"}
  ```
  By default, **driver_name** and **driver_version** are displayed. The display of **driver_path** and **os_user** is controlled by users. For details, see "Application Development Guide > Development Based on JDBC > Connecting to the Database" in *Developer Guide* and "Application Development Guide > Development Based on ODBC > Configuring a Data Source in the Linux OS" in *Developer Guide*.

check_disconnect_query

Parameter description: Specifies whether to terminate the execution of statements on the GaussDB server after the client is disconnected abnormally (for example, socketTimeout is triggered by JDBC, rwtimeout is triggered by libpq and the connection is closed, or the client process is terminated during service running).

Parameter type: Boolean.

Unit: none

Value range:

- **on**: indicates that the GaussDB server stops running the corresponding statements after the client is disconnected unexpectedly.
- **off**: indicates that the GaussDB server does not stop running the corresponding statements after the client is disconnected unexpectedly.

Default value: **on**

Setting method: This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Setting suggestion: Retain the default value.

plat_compat_server_port

Parameter description: Specifies the database configuration item in M-compatible mode. This parameter specifies the TCP port number for protocol listening.

Parameter type: integer.

Unit: none

Value range: 1024 to 65536

Default value: **65536**, indicating that the listening port is not enabled.

Setting method: This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Setting suggestion: The default value is **65536**, indicating that the listening port is disabled. Ports 1024 to 65535 are valid listening ports. Set the parameter to a value of unused port.

14.3.3.2 Security and Authentication (gaussdb.conf)

This section describes parameters about client-to-server authentication.

authentication_timeout

Parameter description: Specifies the timeout period for client authentication. If a client is not authenticated by the server within the period, the server automatically disconnects from the client so that the client does not occupy connection resources.

This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

Value range: an integer ranging from 1 to 600. The unit is s.

Default value: **1min**

auth_iteration_count

Parameter description: Specifies the number of iterations during the generation of encryption information for authentication.

This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

Value range: an integer ranging from 2048 to 134217728.

Default value: **10000**

NOTICE

If the number of iterations is too small, the password storage security is reduced. If the number of iterations is too large, the performance deteriorates in scenarios involving password encryption, such as authentication and user creation. Set the number of iterations based on actual hardware conditions. You are advised to retain the default value.

session_authorization

Parameter description: Specifies the user ID of the current session.

This is a USERSET parameter and can be set only by following the instructions in "SQL Reference > SQL Syntax > SET SESSION AUTHORIZATION" in *Developer Guide*.

Value range: a string.

Default value: **NULL**

session_timeout

Parameter description: Specifies the longest duration allowed when no operations are performed on a client after it is connected to the server.

Parameter type: integer.

Unit: second

Value range: 0 to 86400 (1d).

- The value **0** indicates that the timeout setting is disabled.
- A positive number indicates the maximum duration in which no operation is performed after the connection to the server is set up. When the value of this parameter is exceeded, the client is disconnected from the server.

Default value: 600s

Setting method: This is a USERSET parameter. Set it based on instructions in **Table 14-1**.

Setting suggestion: Retain the default value.

NOTICE

The gsql client of GaussDB has an automatic reconnection mechanism. For local connection of initialized users, the client reconnects to the server if the connection breaks after the timeout.

ssl

Parameter description: Specifies whether to enable the SSL connection on the server. Before using this option, read "Database Quick Start > Connecting to a Database > Using gsql to Connect to a Database" in *Developer Guide*.

This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Value range: Boolean.

- **on** indicates that the SSL connection is enabled on the server. Whether SSL encrypted transmission is used during actual communication depends on the client configuration.
- **off** indicates that the SSL connection is disabled on the server.

NOTICE

To enable this parameter, configure parameters **ssl_cert_file**, **ssl_key_file**, and **ssl_ca_file** as well as the corresponding files. If Chinese cryptographic algorithms are used, ensure that **ssl_enc_cert_file** and **ssl_enc_key_file** are correctly configured. Incorrect configurations may cause database startup failures.

Default value: **on**

require_ssl

Parameter description: Specifies whether SSL is used to connect to the server. This parameter is valid only when **ssl** is set to **on**. Before using this option, read "Database Quick Start > Connecting to a Database > Using gsql to Connect to a Database" in *Developer Guide*.

This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

Value range: Boolean.

- **on** indicates that the server requires SSL connections.
- **off** indicates that the server does not require SSL connections.

NOTICE

GaussDB supports SSL when a client connects to the primary node of the database. It is recommended that the SSL connection be enabled only on the primary node of the database.

Default value: **off**

ssl_ciphers

Parameter description: Specifies the list of encryption algorithms supported by SSL. Only the sysadmin user can access the list.

This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Value range: a string. Separate multiple encryption algorithms by semicolons (;).

Default value: **ALL**

NOTICE

If **ssl_ciphers** is set incorrectly, the database cannot be started properly.

ssl_renegotiation_limit

Parameter description: Specifies the allowed traffic volume over an SSL-encrypted channel before the session key is renegotiated. The renegotiation mechanism reduces the probability that attackers use the password analysis method to crack the key based on a huge amount of data but causes big performance losses. The traffic indicates the sum of transmitted and received traffic. The SSL renegotiation mechanism has been disabled because of potential risks. This parameter is reserved for version compatibility and does not take effect.

This is a USERSET parameter. Set it based on instructions in **Table 14-1**.

Value range: an integer ranging from 0 to 2147483647. The unit is KB. **0** indicates that the renegotiation mechanism is disabled.

Default value: **0**

ssl_cert_file

Parameter description: Specifies the name of the file that contains the SSL server certificate. The path is relative to the data directory.

This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Value range: a string.

Default value: **server.crt**

ssl_key_file

Parameter description: Specifies the name of the file that contains the SSL private key. The path is relative to the data directory.

This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Value range: a string.

Default value: **server.key**

ssl_enc_cert_file

Parameter description: Specifies the name of the SSL server certificate file that is encrypted using Chinese cryptographic algorithms. The path is relative to the data directory.

This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Value range: a string.

Default value: an empty string.

ssl_enc_key_file

Parameter description: Specifies the name of the file that contains the SSL private key. The path is relative to the data directory.

This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Value range: a string.

Default value: an empty string.

ssl_ca_file

Parameter description: Specifies the name of a file that contains CA information. The path is relative to the data directory.

This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Value range: a string. If it is an empty string, no CA file is loaded and client certificate verification is not performed.

Default value: **cacert.pem**

ssl_crl_file

Parameter description: Specifies the certificate revocation list (CRL). If the certificate of a client is in the list, the certificate is invalid. A relative path must be used. The path depends on the data directory.

This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Value range: a string. An empty string indicates that there is no CRL.

Default value: empty

krb_server_keyfile

Parameter description: Specifies the location of the main configuration file of the Kerberos service.

This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

Value range: a string.

Default value: empty

krb_srvname

Parameter description: Specifies the Kerberos service name.

This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

Value range: a string.

Default value: **postgres**

krb_caseins_users

Parameter description: Specifies whether the Kerberos username is case-sensitive.

This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

Value range: Boolean.

- **on** indicates that the Kerberos username is case-insensitive.
- **off** indicates that the Kerberos username is case-sensitive.

Default value: **off**

modify_initial_password

Parameter description: After GaussDB is installed, there is only one initial user account (whose UID is 10) in the database. When a user logs in to the database using this initial account for the first time, this parameter determines whether the password of the initial account needs to be modified.

This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

NOTICE

If the initial user password is not specified during the installation, the initial user password is empty by default after the installation. Before performing other operations, you need to set the initial user password using the gsql client. This parameter no longer takes effect and is reserved only for compatibility with upgrade scenarios.

Value range: Boolean.

- **on** indicates that the password of the initial user needs to be modified upon the first login after the database is successfully installed.
- **off** indicates that the password of the initial user does not need to be modified upon the first login after the database is successfully installed.

Default value: **off**

password_policy

Parameter description: Specifies whether to check the password complexity when you run the **CREATE ROLE/USER** or **ALTER ROLE/USER** command to create or modify the GaussDB account.

This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

NOTICE

For security purposes, do not disable the password complexity policy.

Value range: **0** or **1**

- **0** indicates that no password complexity policy is enabled.
- **1** indicates that the default password complexity policy is enabled.

Default value: **1**

password_reuse_time

Parameter description: Specifies whether to check the reuse interval of the new password when you run the **ALTER USER** or **ALTER ROLE** command to change a user password.

This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

NOTICE

When you change the password, the system checks the values of **password_reuse_time** and **password_reuse_max**.

- If the values of **password_reuse_time** and **password_reuse_max** are both positive numbers, an old password can be reused when it meets either of the reuse restrictions.
- If the value of **password_reuse_time** is **0**, password reuse is restricted based on the number of reuse times, and not on the reuse interval.
- If the value of **password_reuse_max** is **0**, password reuse is restricted based on the reuse interval, and not on the number of reuse times.
- If the values of both **password_reuse_time** and **password_reuse_max** are **0**, password reuse is not restricted.

Value range: a floating-point number ranging from 0 to 3650. The unit is day.

- **0** indicates that the password reuse interval is not checked.
- A positive number indicates that a new password cannot be chosen from passwords in history that are newer than the specified number of days.

Default value: **0**

password_reuse_max

Parameter description: Specifies whether to check the reuse times of the new password when you run the **ALTER USER** or **ALTER ROLE** command to change a user password. Only the sysadmin user can access this parameter.

This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

NOTICE

When you change the password, the system checks the values of **password_reuse_time** and **password_reuse_max**.

- If the values of **password_reuse_time** and **password_reuse_max** are both positive numbers, an old password can be reused when it meets either of the reuse restrictions.
- If the value of **password_reuse_time** is **0**, password reuse is restricted based on the number of reuse times, and not on the reuse interval.
- If the value of **password_reuse_max** is **0**, password reuse is restricted based on the reuse interval, and not on the number of reuse times.
- If the values of both **password_reuse_time** and **password_reuse_max** are **0**, password reuse is not restricted.

Value range: an integer ranging from 0 to 1000.

- **0** indicates that the password reuse times are not checked.
- A positive number indicates that the new password cannot be the one whose reuse times exceed the specified number.

Default value: **0**

password_lock_time

Parameter description: Specifies the duration before a locked account is automatically unlocked.

Parameter type: floating-point.

Unit: day

NOTICE

The locking and unlocking functions take effect only when the values of **password_lock_time** and **failed_login_attempts** are positive numbers.

Value range: 0 to 365. The integer part indicates the number of days, and the decimal part can be converted into hours, minutes, and seconds. For example, **password_lock_time=1.5** indicates one day and 12 hours.

- **0** indicates that an account is not automatically locked if the password verification fails.
- A positive number indicates the duration after which a locked account is automatically unlocked.

Default value: **1**

Setting method: This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

Setting suggestion: Retain the default value. Do not set the automatic unlock time to a large value. Set it to a proper value based on the value of **failed_login_attempts**. If the value of **failed_login_attempts** is too small but the automatic unlock time is too long, the account cannot be used for a long time after the input fails, affecting user experience.

failed_login_attempts

Parameter description: If the number of incorrect password attempts reaches the value of **failed_login_attempts**, the current account is locked. The account is automatically unlocked after the number of seconds specified by **password_lock_time**. During this period, only the sysadmin user can access the account. The automatic account locking policy applies in scenarios such as login and password modification using the **ALTER USER** command.

Parameter type: integer.

Unit: none

NOTICE

The locking and unlocking functions take effect only when the values of **failed_login_attempts** and **password_lock_time** are positive numbers.

Value range: 0 to 1000.

- **0** indicates that the automatic locking function does not take effect.
- A positive number indicates that an account is locked when the number of incorrect password attempts reaches the specified number.

Default value: **10**

Setting method: This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

Setting suggestion: You are advised to set it to a value ranging from 5 to 10. If this parameter is set to a large value that allows too many incorrect password attempts (less than the maximum value of 1000), but no security measure is taken, security risk occurs. If this parameter is set to a small value (greater than 0 and less than 5), the account may be locked due to failed password attempts, affecting normal use.

password_encryption_type

Parameter description: Specifies the encryption type of a user password. Changing the value of this parameter does not automatically trigger the change of the password encryption type of an existing user. Only the password of a new user or the password changed by an existing user is encrypted using the new encryption type.

Parameter type: enumerated type

Unit: none

Value range: **0**, **1**, **2**, or **3**

- **0** indicates that passwords are encrypted with MD5.
- **1** indicates that passwords are encrypted with SHA-256 and MD5.
- **2** indicates that passwords are encrypted with SHA-256.
- **3** indicates that the passwords are encrypted with SM3.

NOTICE

The MD5 encryption algorithm is not recommended because it has lower security and poses security risks.

Default value: 2

Setting method: This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

Setting suggestion: Retain the default value.

password_min_length

Parameter description: Specifies the minimum length of an account password. Only the sysadmin user can access this parameter.

This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

Value range: an integer ranging from 6 to 999.

Default value: **8**

password_max_length

Parameter description: Specifies the maximum length of an account password. Only the sysadmin user can access this parameter.

This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

Value range: an integer ranging from 6 to 999.

Default value: **32**

password_min_uppercase

Parameter description: Specifies the minimum number of uppercase letters that an account password must contain. Only the sysadmin user can access this parameter.

This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

Value range: an integer ranging from 0 to 999

- **0** means no limit.
- An integer ranging from 1 to 999 indicates the minimum number of uppercase letters required in a password when you create an account.

Default value: **0**

password_min_lowercase

Parameter description: Specifies the minimum number of lowercase letters that an account password must contain. Only the sysadmin user can access this parameter.

This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

Value range: an integer ranging from 0 to 999

- **0** means no limit.
- An integer ranging from 1 to 999 indicates the minimum number of lowercase letters required in a password when you create an account.

Default value: **0**

password_min_digital

Parameter description: Specifies the minimum number of digits that an account password must contain. Only the sysadmin user can access this parameter.

This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

Value range: an integer ranging from 0 to 999

- **0** means no limit.

- An integer ranging from 1 to 999 indicates the minimum number of digits required in a password when you create an account.

Default value: **0**

password_min_special

Parameter description: Specifies the minimum number of special characters that an account password must contain. Only the sysadmin user can access this parameter.

This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

Value range: an integer ranging from 0 to 999

- **0** means no limit.
- An integer ranging from 1 to 999 indicates the minimum number of special characters required in a password when you create an account.

Default value: **0**

password_effect_time

Parameter description: Specifies the validity period of an account password.

Parameter type: floating-point.

Unit: day

Value range: 0 to 999. The integer part indicates the number of days, and the decimal part can be converted into hours, minutes, and seconds. For example, **password_lock_time=0.5** indicates 0 days and 12 hours.

- **0** indicates that the validity period restriction is disabled.
- A floating-point number from 1 to 999 indicates the number of days for which an account password is valid. When the password is about to expire or has expired, the system prompts the user to change the password.

Default value: **0**

Setting method: This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

Setting suggestion: For security purposes, you are advised to retain the default value.

password_notify_time

Parameter description: Specifies how many days in advance a user is notified before a password expires.

This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

Value range: an integer ranging from 0 to 999. The unit is day.

- **0** indicates that the reminder is disabled.
- An integer ranging from 1 to 999 indicates the number of days prior to password expiration that a user will receive a reminder.

Default value: **7**

enable_innertool_cert

Parameter description: Specifies whether internal tools use certificate-based authentication.

This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

Value range: Boolean.

- **on**: indicates that internal tools use certificate-based authentication.
- **off**: indicates that internal tools do not use certificate-based authentication.

Default value: **off**

NOTICE

- This parameter takes effect only when both this parameter and **ssl** are enabled.
- By default, the certificate in the *$GAUSSHOME*/**share/sslcert/gsql** directory is used. The **Common Name** of the certificate must be the same as the initial user. Otherwise, the authentication fails.
- After this parameter takes effect, the initial user uses certificate-based authentication, and other users use password-based authentication.
- If this parameter is enabled, the initial user can remotely connect to the database using certificate-based authentication.

ssl_cert_notify_time

Parameter description: Specifies the number of days prior to SSL server certificate expiration that a user will receive a reminder. When the SSL certificate is initialized during connection establishment, if the duration from the current time to the certificate expiration time is shorter than the specified value, an expiration notification is recorded in the log.

This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

Value range: an integer ranging from 7 to 180. The unit is day.

Default value: **90**

plat_compat_allow_public_key_retrieval

Parameter description: Specifies the database configuration item in M-compatible mode. This parameter specifies whether to enable the RSA public key function on the client. By default, this function is disabled.

Parameter type: Boolean.

- **on**: The kernel allows the client to request the RSA public key for password transmission encryption.
- **off**: The client is not allowed to request the RSA public key.

Unit: none

Default value: **off**

Setting method: This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Setting suggestions: If the RSA connection is used, set this parameter to **on** and SSL to **off**.

14.3.3.3 Communications Library Parameters

This section describes parameter settings and value ranges for communications libraries.

tcp_keepalives_idle

Parameter description: Specifies the interval for transmitting keepalive signals on an OS that supports the **TCP_KEEPIDLE** socket option. If no keepalive signal is transmitted, the connection is in idle mode.

This is a USERSET parameter. Set it based on instructions in **Table 14-1**.

NOTICE

- If the OS does not support **TCP_KEEPIDLE**, set this parameter to **0**.
- The parameter is ignored on an OS where connections are established using the UDS.
- If this parameter is set to **0**, the system value is used.
- This parameter is not shared among different sessions. That is, different session connections may have different values.
- The parameter value in the current session connection, not the value of the GUC copy, is displayed.

Value range: 0 to 3600. The unit is s.

Default value: **60**

tcp_keepalives_interval

Parameter description: Specifies the response time before retransmission on an OS that supports the **TCP_KEEPINTVL** socket option.

This is a USERSET parameter. Set it based on instructions in **Table 14-1**.

Value range: 0 to 180. The unit is s.

Default value: **30**

NOTICE

- If the OS does not support **TCP_KEEPINTVL**, set this parameter to **0**.
- The parameter is ignored on an OS where connections are established using the UDS.
- If this parameter is set to **0**, the system value is used.
- This parameter is not shared among different sessions. That is, different session connections may have different values.
- The parameter value in the current session connection, not the value of the GUC copy, is displayed.

tcp_keepalives_count

Parameter description: Specifies the number of keepalive signals that can be waited before the GaussDB server is disconnected from the client on an OS that supports the **TCP_KEEPCNT** socket option.

This is a USERSET parameter. Set it based on instructions in **Table 14-1**.

NOTICE

- If the OS does not support **TCP_KEEPCNT**, set this parameter to **0**.
- The parameter is ignored on an OS where connections are established using the UDS.
- If this parameter is set to **0**, the system value is used.
- This parameter is not shared among different sessions. That is, different session connections may have different values.
- The parameter value in the current session connection, not the value of the GUC copy, is displayed.

Value range: 0 to 100. **0** indicates that the connection is immediately broken if GaussDB does not receive a keepalive signal from the client.

Default value: 20

tcp_user_timeout

Parameter description: Specifies the maximum duration for which the transmitted data can remain in the unacknowledged state before the TCP connection is forcibly closed when the GaussDB sends data on the OS that supports the **TCP_USER_TIMEOUT** socket option.

This is a SIGHUP parameter. Set it based on instructions in **Table 14-1**.

NOTICE

- If the OS does not support the **TCP_USER_TIMEOUT** option, the value of this parameter does not take effect. The default value is **0**.
- The parameter is ignored on an OS where connections are established using the UDS.

Value Range: 0 to 3600000. The unit is ms. The value **0** indicates that the value is set based on the OS.

Default value: **0**

Note that the effective result of this parameter varies according to the OS kernel.

- For AArch64 EulerOS (Linux kernel version: 4.19), the timeout interval is the value of this parameter.
- For x86 Euler 2.5 (Linux kernel version: 3.10), the timeout interval is not the value of this parameter but the maximum value in different ranges. That is, the timeout interval is the maximum upper limit of the total Linux TCP retransmission duration to which the value of **tcp_user_timeout** belongs. For example, if **tcp_user_timeout** is set to **40000**, the total retransmission duration is 51 seconds.

  Table 14-4 Value of tcp_user_timeout for x86 Euler 2.5 (Linux kernel version: 3.10)

Number of Linux TCP Retransmission Times	Total Linux TCP Retransmission Duration Range (s)	Example of tcp_user_timeout (ms)	Actual Linux TCP Retransmission Duration (s)
1	(0.2,0.6]	400	0.6
2	(0.6,1.4]	1000	1.4
3	(1.4,3]	2000	3
4	(3,6.2]	4000	6.2
5	(6.2,12.6]	10000	12.6
6	(12.6,25.4]	20000	25.4
7	(25.4,51]	40000	51
8	(51,102.2]	80000	102.2
9	(102.2,204.6]	150000	204.6
10	(204.6,324.6]	260000	324.6
11	(324.6,444.6]	400000	444.6

Note: The duration of each TCP retransmission increases exponentially with the number of retransmission times. When the duration of a TCP retransmission reaches 120 seconds, the duration of each subsequent retransmission does not change.

comm_proxy_attr

Parameter description: Specifies the parameters related to the communications proxy library.

NOTE

- This parameter applies only to the centralized Arm standalone system running EulerOS 2.9.
- This function takes effect when the thread pool is enabled, that is, **enable_thread_pool** is set to **on**.
- When setting this parameter, you need to set the GUC parameter **local_bind_address** to the IP address of the NIC of the **libos_kni**.
- **Parameter template**: comm_proxy_attr = '{enable_libnet:true, enable_dfx:false, numa_num:4, numa_bind:[[30,31],[62,63],[94,95],[126,127]]}'
- Parameters that need to be configured include:
  - **enable_libnet**: specifies whether to enable the user-mode protocol. The value can be **true** or **false**.
  - **enable_dfx**: specifies whether to enable the communications proxy library view. The value can be **true** or **false**.
  - **numa_num**: number of NUMA nodes in the system. 2P and 4P servers are supported. The value can be **4** or **8**.
  - **numa_bind**: core binding parameter of the agent thread. Each numa has two CPUs. There are a total of **numa_num** groups. The value range is [0,*Number of CPUs* – 1].

This is a POSTMASTER parameter. Set it based on instructions in **Table 14-1**.

Value range: a string, consisting of one or more characters.

Default value: **'none'**

umdk_enabled

Parameter description: Specifies whether UMDK is enabled for the current primary and standby DNs in the database. If the UMDK protocol is used for communication between the primary and standby DNs, the related log keyword on DNs is umdk. If the TCP protocol is used for communication between the primary and standby DNs, logs are recorded. Currently, this function is supported only in some scenarios.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: UMDK is enabled. If the current database instance does not support the UMDK network communication protocol, TCP is used for communication between the primary and standby DNs. Otherwise, UMDK is used for communication between the primary and standby DNs.

- **off**: The UMDK function is disabled. TCP is used for communication between the primary and standby DNs.

Default value: **off**

Setting method: This is a POSTMASTER parameter. Set it based on instructions in Table 14-1.

Setting suggestion: Set this parameter based on the actual application scenarios.

NOTE

- GaussDB adapts to the UMDK communication protocol stack. The UMDK capability is provided and maintained by the OS.
- By default, the **umdk_enabled** parameter is disabled in GaussDB. After this parameter is enabled, whether GaussDB supports this capability depends on whether the OS provides and supports the UMDK function.
- The UMDK capability depends on the hardware NIC and software OS configuration. For example, the hardware NIC should be 1823V200 and the OS should be HCE2.0.
- To check whether the environment has the UMDK capability, run the **lsmod | grep ums** statement as the **root** user to check whether the UMS module is loaded to the operating systemOS.

  If the result of the preceding operation shows that the current environment does not support the UMDK (UMS module) or the UMS module is faulty, enabling **umdk_enabled** will cause the connection between the primary and standby nodes to fail. Before enabling **umdk_enabled**, ensure that the environment has the UMDK capability.

umdk_port

Parameter description: Specifies the listening port of the network communication link between the primary and standby DNs when the UMDK protocol is used for communication between the primary and standby DNs.

Parameter type: integer.

Unit: none

Value range: 1 to 65535

Default value: ***dataPortBase* + 15**. The value of *dataPortBase* is the same as the value of GUC parameter **port** set when the database is installed for the first time.

Setting method: This is a POSTMASTER parameter. Set it based on instructions in Table 14-1.

Setting suggestion: Retain the default value.

NOTE

- GaussDB adapts to the UMDK communication protocol stack. The UMDK capability is provided and maintained by the OS.
- By default, GaussDB does not listen on the port specified by **umdk_port**. The UMDK protocol function of the database is enabled and related ports are listened on only when the GUC parameter **umdk_enabled** is enabled and the hardware and software in the current environment support the UMDK protocol. Then, the primary and standby DNs start to execute the UMS network protocol (a type of UMDK network protocols).

14.3.4 Resource Consumption

14.3.4.1 Memory

This section describes memory parameters.

NOTICE

These parameters, except **local_syscache_threshold**, take effect only after the database restarts.

memorypool_enable

Parameter description: Specifies whether to enable a memory pool.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the memory pool is enabled.
- **off** indicates that the memory pool is disabled.

Default value: **off**

memorypool_size

Parameter description: Specifies the memory pool size.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 128 x 1024 to *INT_MAX*/2. The unit is KB.

Default value: **512MB**

enable_memory_limit

Parameter description: Specifies whether to enable the logical memory management module.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the logical memory management module is enabled.
- **off** indicates that the logical memory management module is disabled.

Default value: **on**

CAUTION

- Fixed overhead exists, that is, **shared_buffers** and metadata (about 200 MB). If **max_process_memory** minus the fixed overhead is less than 2 GB, GaussDB forcibly sets **enable_memory_limit** to **off**. Metadata is the memory used in GaussDB and is related to some concurrent parameters, such as **max_connections**, **thread_pool_attr**, and **max_prepared_transactions**.
- If this parameter is set to **off**, the memory used by the database is not limited. When a large number of concurrent or complex queries are performed, too much memory is used, which may cause OS OOM problems.

max_process_memory

Parameter description: Specifies the maximum physical memory of a database node.

Parameter type: integer.

Unit: KB

Value range: 2097152 to 2147483647

Default value:

1400GB (196-core CPU/1536 GB memory); **900GB** (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory); **680GB** (96-core CPU/768 GB memory); **560GB** (80-core CPU/640 GB memory); **450GB** (64-core CPU/512 GB memory); **420GB** (60-core CPU/480 GB memory); **200GB** (32-core CPU/256 GB memory); **90GB** (16-core CPU/128 GB memory); **40GB** (8-core CPU/64 GB memory); **20GB** (4-core CPU/32 GB memory); **10GB** (4-core CPU/16 GB memory)

CAUTION

If this parameter is set to a value greater than the physical memory of the server, the OS OOM problem may occur.

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: This parameter is used to prevent node OOM caused by memory bloat, ensuring system reliability. The value of this parameter on the database node depends on the physical memory of the system and the number of primary database nodes deployed on a single node. The recommended formula is as follows: (Physical memory size – **vm.min_free_kbytes**) x 0.7/Number of primary nodes. **vm.min_free_kbytes** in this formula indicates that the OS memory reserved for the kernel to receive and send data. Its value is at least 5% of the total memory. That is, **max_process_memory** = Physical memory size x 0.665/Number of primary nodes.

enable_memory_context_control

Parameter description: Enables the function of checking whether the number of memory contexts exceeds the specified limit. This parameter applies only to the DEBUG version.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the function of checking the number of memory contexts is enabled.
- **off** indicates that the function of checking the number of memory contexts is disabled.

Default value: **off**

uncontrolled_memory_context

Parameter description: Specifies which memory context will not be checked when the function of checking whether the number of memory contexts exceeds the specified limit is enabled. This parameter applies only to the DEBUG version.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

During the query, the title meaning string "MmgrMemoryController white list:" is added to the beginning of the parameter value.

Value range: a string.

Default value: empty

shared_buffers

Parameter description: Specifies the size of shared memory used by GaussDB. Increasing the value of this parameter causes GaussDB to request more System V shared memory than the default configuration allows.

Parameter type: integer.

Unit: page (8 KB)

Value range: 16 to 1073741823. The value of this parameter must be an integer multiple of **BLCKSZ**. Currently, **BLCKSZ** is set to **8KB**. That is, the value of this parameter must be an integer multiple of 8 KB.

Default value:

560GB (196-core CPU/1536 GB memory); **360GB** (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory); **270GB** (96-core CPU/768 GB memory); **220GB** (80-core CPU/640 GB memory); **180GB** (64-core CPU/512 GB memory); **160GB** (60-core CPU/480 GB memory); **80GB** (32-core CPU/256 GB memory); **36GB** (16-core CPU/128 GB memory); **16GB** (8-core CPU/64 GB memory); **8GB** (4-core CPU/32 GB memory); **2GB** (4-core CPU/16 GB memory)

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion:

1. Set **shared_buffers** to a value less than 40% of the memory.
2. If **shared_buffers** is set to a larger value, increase the value of **checkpoint_segments** because a longer period of time is required to write a large amount of new or changed data.
3. If the process fails to be restarted after the value of **shared_buffers** is changed, perform either of the following operations based on the error information:
   a. Adjust the **kernel.shmall**, **kernel.shmmax**, and **kernel.shmmin** OS parameters. For details, see "Configuring Other OS Parameters" in *Installation Guide*.
   b. Run the **free -g** command to check whether the available memory and swap space of the OS are sufficient. If the memory is insufficient, manually stop other user programs that occupy much memory.
   c. Set this parameter to the recommended default value for different specifications. You are advised not to change the value of **shared_buffers**. Otherwise, it may be too large or too small. The following condition must be met: **data_replicate_buffer_size** + **segment_buffers** + **shared_buffers** + **wal_buffers** + **temp_buffers** + **maintenance_work_mem** + **work_mem** + **query_mem** + (Standby node) **wal_receiver_buffer_size** < **max_process_memory** < Memory size of the physical machine. If the value of the memory parameter is too large and exceeds the upper limit of the physical memory, the database cannot be started because it cannot allocate sufficient memory.

page_version_check

Parameter description: Specifies whether to perform verification for underlying storage faults and pages not marked as dirty based on page version information. **page_version_check** is a level-3 switch. The verification for underlying storage faults is to check whether a page read from the underlying storage is of a correct version, which prevents loss of page version information caused by a fault such as a disk power failure. The verification for pages not marked as dirty is to check whether modified pages are not marked as dirty, and is controlled by an independent switch **page_missing_dirty_check**.

Value type: enumerated type

Unit: none

Value range:

- **off**: The verification for underlying storage faults and pages not marked as dirty is disabled.
- **memory**: The page version verification function (that is, verification for underlying storage faults and pages not marked as dirty) in pure memory mode is enabled. The page version information is cached only in the memory and will be lost after a restart.
- **persistence**: The persistent page version verification function (that is, verification for underlying storage faults and pages not marked as dirty) is enabled. The page version information is persisted to files and will not be lost after a restart.

Default value: **memory**

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestions: Set the value of this parameter based on different specifications, that is, **off** (four-core CPU/16 GB memory, four-core CPU/32 GB memory, and eight-core CPU/64 GB memory) or **memory** (16-core CPU/128 GB memory, 32-core CPU/256 GB memory, 60-core CPU/480 GB memory, 64-core CPU/512 GB memory, 72-core CPU/576 GB memory, 80-core CPU/640 GB memory, and 96-core CPU/768 GB memory, 96-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 128-core CPU/1024 GB memory, and 196-core CPU/1536 GB memory). Setting this parameter to **memory** affects the performance of a device, and the one with smaller specifications suffers more. (For example, the performance of a 16-core CPU/128 GB memory device using TPC-C model is about 7%.) If the system needs to restart frequently, you are advised to set this parameter to **persistence** to ensure that the version information on the page is not lost. However, the performance will be affected.

page_missing_dirty_check

Parameter description: Checks whether the modified pages are not marked as dirty. **page_missing_dirty_check** is controlled by **page_version_check**. If **page_version_check** is set to **off**, setting **page_missing_dirty_check** to **on** does not take effect.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: The verification for pages not marked as dirty is performed.
- **off**: The verification for pages not marked as dirty is not performed.

Default value: **off**

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: You are advised to enable this function in test scenarios to detect as many pages not marked as dirty that lead to code bugs in non-production environments as possible. On the live network, this function is disabled by default to avoid extra overhead and performance deterioration.

page_version_max_num

Parameter description: Specifies the maximum number of page versions that can be cached in the memory. This parameter is valid only when **page_version_check** is not set to **off**. The proper number must be twice to four times the value of **shared_buffers**. Each **page_version** occupies 36 bytes of memory. Pay attention to the memory usage.

Parameter type: integer.

Unit: none

Value range: 0 to 2147483647.

- **0**: When **page_version_check** is not set to **OFF**, the value of **page_version_max_num** is automatically calculated based on the value of **shared_buffers** using the following formula: **shared_buffers** x 2. For example, 32 MB of **shared_buffers** corresponds to 4096 buffers. Therefore, the value of this parameter is set to **8192**.
- Non-zero values: The manually configured value is forcibly used.
- If **page_version_check** is not set to **OFF**, the value cannot be less than 16 times of **page_version_partitions**. Otherwise, it is forcibly set to a value equal to **page_version_partitions** x 16.

Default value: **0**

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: If high performance is required and the memory is sufficient, you are advised to manually set this parameter to four times the value of **shared_buffers** and the ratio of this parameter to **page_version_partitions** is [256, 1024].

page_version_partitions

Parameter description: Specifies the number of hash table partitions in cached page version information in the memory. This parameter directly affects the hash query efficiency and hash conflict probability.

Parameter type: integer.

Unit: none

Value range: 0–2097152

- **0**: **When page_version_check** is not set to **OFF**, the value is automatically calculated based on **page_version_max_num**, that is, **page_version_partitions** = **page_version_max_num**/512. If the automatically calculated value is smaller than 4, the parameter is forcibly set to **4**.
- Non-zero values: The manually configured value is forcibly used. If **page_version_check** is not set to **OFF,** the minimum value is **4**. If the value is less than 4, the parameter is forcibly set to **4**.

Default value: **0**

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: If you have high performance requirements, you are advised to manually set this parameter to a value 1/256 to 1/1024 of the value of **page_version_max_num**.

page_version_recycler_thread_num

Parameter description: Specifies the number of threads for recycling and verifying page version information. This parameter is valid only when **page_version_check** is not set to **off**.

Parameter type: integer.

Unit: none

Value range: 0–16

- If **page_version_check** is set to **memory**:
  - **0**: The value is automatically calculated based on the value of **page_version_partitions** using the following formula: **page_version_recycler_thread_num** = **page_version_partitions**/16384. If the automatically calculated value is greater than 4, the parameter is forcibly set to **4**.
  - Non-zero values: The manually configured value is forcibly used.
  - The parameter cannot be set to a value greater than that of **page_version_partitions**. Otherwise, **page_version_partitions** is forcibly set to a value equal to that of **page_version_partitions**.
- If **page_version_check** is set to **persistence**:

  If the value is less than 2, set this parameter to **2**. If the value is greater than or equal to 2, the manually configured parameter value is forcibly used.

Default value: **0**

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value **0**.

verify_log_buffers

Parameter description: Specifies the size of the verifyLog buffer. This parameter is valid only when **page_version_check** is set to **persistence**. The verifyLog buffer memory is managed by page, and each page is 8 KB.

Parameter type: integer.

Unit: page (8 KB)

Value range: 4-262144

Default value: **4** (32KB)

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**. For example, the value **131072** of **verify_log_buffers** indicates that the size is 131072 x 8 KB = 1GB, and the value **131072kB** indicates that **verify_log_buffers** is 131072 KB. If the value contains a unit, the value must be KB, MB, or GB and must be an integer multiple of 8 KB.

Setting suggestion: Set this parameter based on the system hardware specifications.

1GB (196-core CPU/1536 GB memory, 128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 64-core CPU/512 GB memory, 60-core CPU/480 GB memory, 32-core CPU/256 GB memory); **512MB** (16-core CPU/128 GB memory); **256MB** (8-core CPU/64 GB memory); **128MB** (4-core CPU/32 GB memory); **16MB** (4-core CPU/16 GB memory)

segment_buffers

Parameter description: Specifies the memory size of a GaussDB segment-page metadata page.

Parameter type: integer.

Unit: KB

Value range: 16 to 1073741823. The value of this parameter must be an integer multiple of **BLCKSZ**. Currently, **BLCKSZ** is set to **8KB**. That is, the value of this parameter must be an integer multiple of 8 KB.

Default value:

8MB (196-core CPU/1536 GB memory, 128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 64-core CPU/512 GB memory, 60-core CPU/480 GB memory, 32-core CPU/256 GB memory, 16-core CPU/128 GB memory, 8-core CPU/64 GB memory, 4-core CPU/32 GB memory); **128KB** (4-core CPU/16 GB memory)

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestions: **segment_buffers** is used to cache the content of segment-page headers, which is key metadata information. To improve performance, it is recommended that the segment headers of ordinary tables be cached in the buffer and not be replaced. You are advised to set this parameter based on the following formula: Number of tables (including indexes and TOAST tables) x Number of partitions x 3 + 128. Multiplying by 3 is because each table (partition) has some extra metadata segments. Generally, a table has three segments. Adding 128 at last is because segment-page tablespace management requires a certain number of buffers. If this parameter is set to a small value, it takes a long time to create a segment-page table for the first time. Therefore, you are advised to retain the default value to avoid setting **segment_buffers** to an excessively large or small value. The following condition must be met: **data_replicate_buffer_size** + **segment_buffers** + **shared_buffers** + **wal_buffers** + **temp_buffers** + **maintenance_work_mem** + **work_mem** + **query_mem** + (Standby node) **wal_receiver_buffer_size** < **max_process_memory** < Memory size of the physical machine. If the value of the memory parameter is too large and exceeds the upper limit of the physical memory, the database cannot be started because it cannot allocate sufficient memory.

bulk_write_ring_size

Parameter description: Specifies the size of the ring buffer used by the operation when a large amount of data is written (for example, the copy operation).

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 16384 to 2147483647. The unit is KB.

Default value: **2GB**

Setting suggestion: Increase the value of this parameter on database nodes if a huge amount of data will be imported.

standby_shared_buffers_fraction

Parameter description: Specifies the **shared_buffers** proportion used on the server where a standby instance is deployed.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a double-precision floating-point number ranging from 0.1 to 1.0

Default value: **1**

temp_buffers

Parameter description: Specifies the maximum size of local temporary buffers used by a database session.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

temp_buffers can be modified only before the first use of temporary tables within each session. Subsequent attempts to change the value of this parameter will not take effect on that session.

A session allocates temporary buffers based on the value of **temp_buffers**. If a large value is set in a session that does not require many temporary buffers, only the overhead of one buffer descriptor is added. If a buffer is used, additional 8192 bytes will be consumed for it.

Value range: an integer ranging from 100 to 1073741823. The unit is 8 KB.

Default value: **1MB**

max_prepared_transactions

Parameter description: Sets the maximum number of transactions that can be in the "prepared" state simultaneously. Increasing the value of this parameter causes GaussDB to request more System V shared memory than the default configuration allows.

When GaussDB is deployed as an HA system, set this parameter on standby nodes to a value greater than or equal to that on primary nodes. Otherwise, queries will fail on the standby nodes.

Parameter type: integer.

Unit: none

Value range: 0 to 262143

Default value:

200 (196-core CPU/1536 GB memory, 128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 64-core CPU/512 GB memory, 60-core CPU/480 GB memory, 32-core CPU/256 GB memory, 16-core CPU/128 GB memory, 8-core CPU/64 GB memory, 4-core CPU/32 GB memory); **0** (4-core CPU/16 GB memory)

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestions: Generally, explicit PREPARE operations are not required for transactions. If explicit PREPARE operations are performed for transactions, set this parameter to a value greater than the number of concurrent services that require PREPARE to prevent preparation failures.

work_mem

Parameter description: Specifies the amount of memory to be used by internal sort operations and hash tables before they write data into temporary disk files. Sort operations are required for **ORDER BY**, **DISTINCT**, and merge joins. Hash tables are used in hash joins, hash-based aggregation, and hash-based processing of IN subqueries.

In a complex query, several sort or hash operations may run in parallel; each operation will be allowed to use as much memory as this parameter specifies. If the memory is insufficient, data will be written into temporary files. In addition, several running sessions could be performing such operations concurrently. Therefore, the total memory used may be many times the value of **work_mem**.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 64 to 2147483647. The unit is KB.

Default value:

280MB (196-core CPU/1536 GB memory); **256MB** (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory); **128MB** (80-core CPU/640 GB memory, 64-core CPU/512 GB memory, 60-core CPU/480 GB memory, 32-core CPU/256 GB memory, 16-core CPU/128 GB memory); **64MB** (8-core CPU/64 GB memory) ; **32MB** (4-core CPU/32 GB memory); **16MB** (4-core CPU/16 GB memory)

NOTICE

Setting suggestion:

If the physical memory specified by **work_mem** is insufficient, additional operator calculation data will be written into temporary tables based on query characteristics and the degree of parallelism. This reduces performance by five to ten times, and prolongs the query response time from seconds to minutes.

- For complex serial queries, each query requires five to ten associated operations. Set **work_mem** using the following formula: **work_mem** = 50% of the memory/10.
- For simple serial queries, each query requires two to five associated operations. Set **work_mem** using the following formula: **work_mem** = 50% of the memory/5.
- For concurrent queries, set **work_mem** using the following formula: **work_mem** = **work_mem** for serial queries/Number of concurrent SQL statements.
- BitmapScan hash tables are also restricted by **work_mem**, but will not be forcibly flushed to disks. In the case of complete lossify, every 1 MB memory occupied by the hash table corresponds to a 16 GB page of BitmapHeapScan (32 GB for Ustore). After the upper limit of **work_mem** is reached, the memory increases linearly with the data access traffic based on this ratio.

query_mem

Parameter description: Specifies the memory used by a query.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: 0 or an integer greater than 32 MB. The default unit is KB.

Default value: **0**

NOTICE

- If the value of **query_mem** is greater than 0, the optimizer adjusts the memory cost estimate to this value when generating an execution plan.
- If the value is set to a negative value or a positive integer less than 32 MB, the default value **0** is used. In this case, the optimizer does not adjust the estimated query memory.

query_max_mem

Parameter description: Specifies the maximum memory that can be used by a query.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: 0 or an integer greater than 32 MB. The default unit is KB.

Default value: **0**

NOTICE

- If the value of **query_max_mem** is greater than 0, an error is reported when the query memory usage exceeds the value.
- If the value is set to a negative value or a positive integer less than 32 MB, the default value **0** is used. In this case, the optimizer does not limit the query memory.

maintenance_work_mem

Parameter description: Specifies the maximum amount of memory to be used by maintenance operations, such as VACUUM, CREATE INDEX, and ALTER TABLE ADD FOREIGN KEY. This parameter may affect the execution efficiency of **VACUUM**, **VACUUM FULL**, **CLUSTER**, and **CREATE INDEX**.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1024 to *INT_MAX*. The unit is KB.

Default value:

2GB (196-core CPU/1536 GB memory, 128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 64-core CPU/512 GB memory, 60-core

CPU/480 GB memory); **1GB** (32-core CPU/256 GB memory); **512MB** (16-core CPU/128 GB memory); **256MB** (8-core CPU/64 GB memory); **128MB** (4-core CPU/32 GB memory); **64MB** (4-core CPU/16 GB memory)

NOTICE

Setting suggestion:

- The value of this parameter must be greater than that of **work_mem** so that database dumps can be more quickly cleared or restored. In a database session, only one maintenance operation can be performed at a time. Maintenance is usually performed when there are not many running sessions.
- When the **Autovacuum** process is running, up to ***autovacuum_max_workers*** times this memory may be allocated. In this case, set **maintenance_work_mem** to a value greater than or equal to that of **work_mem**.
- If a large amount of data is to be clustered, increase the value of this parameter in the session.

max_stack_depth

Parameter description: Specifies the maximum safe depth of the GaussDB execution stack. The safety margin is required because the stack depth is not checked in every routine in the server, but only in key potentially-recursive routines, such as expression evaluation.

Parameter type: integer.

Unit: KB

Value range: [100,*INT_MAX*]

Default value:

- If the value of **ulimit -s** minus 640 KB is greater than or equal to 2 MB, the default value of this parameter is **2 MB**.
- If the value of **ulimit -s** minus 640 KB is less than 2 MB, the default value of this parameter is the value of **ulimit -s** minus 640 KB.

Setting method: This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

When setting this parameter, comply with the following principles:

- The database needs to reserve 640 KB stack depth. Therefore, the ideal value of this parameter is the actual stack size limit enforced by the OS kernel (as set by **ulimit -s**) minus 640 KB.
- If the value of this parameter is greater than the value of **ulimit -s** minus 640 KB before the database is started, the database fails to be started. During database running, if the value of this parameter is greater than the value of **ulimit -s** minus 640 KB, this parameter does not take effect.
- If the value of **ulimit -s** minus 640 KB is less than the minimum value of this parameter, the database fails to be started.
- Setting this parameter to a value greater than the actual kernel limit means that a running recursive function may crash an individual backend process.
- Since not all OSs provide this function, you are advised to set a specific value for this parameter.
- The default value is **2 MB**, which is relatively small and does not easily cause system breakdown.

bulk_read_ring_size

Parameter description: Specifies the size of the ring buffer used by the operation when a large amount of data is queried (for example, during large table scanning).

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 256 to 2147483647. The unit is KB.

Default value: **16MB**

enable_early_free

Parameter description: Specifies whether the operator memory can be released in advance.

Parameter type: Boolean.

Unit: none

Value range:

- **on** indicates that the operator memory can be released in advance.
- **off** indicates that the operator memory cannot be released in advance.

Default value: **on**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

local_syscache_threshold

Parameter description: Specifies the size of system catalog cache in a session. If **enable_global_plancache** is enabled, **local_syscache_threshold** does not take effect when it is set to a value less than 16 MB to ensure that GPC takes effect. The minimum value is 16 MB. If **enable_global_syscache** and **enable_thread_pool** are enabled, this parameter indicates the total cache size of the current thread and sessions bound to the current thread.

Parameter type: integer.

Unit: kB

Value range:

Method 1: Set this parameter to an integer without a unit. The integer ranges from 1 x 1024 to 512 x 1024. You are advised to set this parameter to an integer multiple of 1024. For example, the value **2048** indicates 2048 KB.

Method 2: Set this parameter to a value with a unit. The value ranges from 1 x 1024 KB to 512 x 1024 KB. For example, the value **32MB** indicates 32 MB. The unit can only be kB, MB, or GB.

Default value:

32MB (196-core CPU/1536 GB memory); **16MB** (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 64-core CPU/512 GB memory, 60-core CPU/480 GB memory, 32-core CPU/256 GB memory, 16-core CPU/128 GB memory, 8-core CPU/64 GB memory, 4-core CPU/32 GB memory, 4-core CPU/16 GB memory)

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

memory_trace_level

Parameter description: Specifies the control level for recording memory allocation information after the dynamic memory usage exceeds 90% of the maximum dynamic memory. This parameter takes effect only when **use_workload_manager** and **enable_memory_limit** are enabled. This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: enumerated values

- **none**: indicates that memory application information is not recorded.
- **level1**: After the dynamic memory usage exceeds 90% of the maximum dynamic memory, the following memory information is recorded and saved in the *$GAUSSLOG*/**mem_log** directory:
  - Global memory overview.
  - Memory usage of the top 20 memory contexts of the instance, session, and thread types.
  - The **totalsize** and **freesize** columns for each memory context.

- **level2**: After the dynamic memory usage exceeds 90% of the maximum dynamic memory, the following memory information is recorded and saved in the *$GAUSSLOG*/**mem_log** directory:
  - Global memory overview.
  - Memory usage of the top 20 memory contexts of the instance, session, and thread types.
  - The **totalsize** and **freesize** columns for each memory context.
  - Detailed information about all memory applications in each memory context, including the file where the allocated memory is located, line number, and size.

Default value: **level1**

NOTICE

- If this parameter is set to **level2**, the memory allocation details (file, line, and size) of each memory context are recorded, which greatly affects the performance. Therefore, exercise caution when setting this parameter.
- You can use the system function gs_get_history_memory_detail(cstring) to query the recorded memory snapshot information. For details about the function, see "SQL Reference > Functions and Operators > Statistics Functions" in *Developer Guide*.
- The recorded memory context is obtained after all memory contexts of the same type with the same name are summarized.

resilience_memory_reject_percent

Parameter description: Specifies the dynamic memory usage percentage for escape from memory overload. This parameter takes effect only when the GUC parameters **use_workload_manager** and **enable_memory_limit** are enabled. This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string, consisting of one or more characters.

This parameter consists of **recover_memory_percent** and **overload_memory_percent**. The meanings of the two parts are as follows:

- **recover_memory_percent**: Percentage of the dynamic memory usage when the memory recovers from overload to the maximum dynamic memory. When the dynamic memory usage is less than the maximum dynamic memory multiplied by the value of this parameter, the overload escape function is disabled and new connections are allowed. The value ranges from 0 to 100. The value indicates a percentage.
- **overload_memory_percent**: Percentage of the dynamic memory usage to the maximum dynamic memory when the memory is overloaded. When the dynamic memory usage is greater than the maximum dynamic memory multiplied by the value of this parameter, the current memory is overloaded. In this case, the overload escape function is triggered to kill sessions and new connections are prohibited. The value ranges from 0 to 100. The value indicates a percentage.

Default value: **'0,0'**, indicating that the escape from memory overload function is disabled.

Example:

```
resilience_memory_reject_percent = '70,90'
```

When the memory usage exceeds 90% of the upper limit, new connections are forbidden and stacked sessions are killed. When the memory usage is less than 70% of the upper limit, session killing is stopped and new connections are allowed.

NOTICE

- You can query the maximum dynamic memory and used dynamic memory in the gs_total_memory_detail view. **max_dynamic_memory** indicates the maximum dynamic memory, and **dynamic_used_memory** indicates the used dynamic memory.
- If this parameter is set to a small value, the escape from memory overload process is frequently triggered. As a result, ongoing sessions are forcibly logged out, and new connections fail to be connected for a short period of time. Therefore, exercise caution when setting this parameter based on the actual memory usage.
- The values of **recover_memory_percent** and **overload_memory_percent** can be **0** at the same time. In addition, the value of **recover_memory_percent** must be smaller than that of **overload_memory_percent**. Otherwise, the setting does not take effect.

resilience_escape_user_permissions

Parameter description: Specifies the escape permission of users. You can set it for multiple users and separate users by commas (,). The value **sysadmin** indicates that jobs of the sysadmin user can be canceled by the escape function. The value **monadmin** indicates that jobs of the monadmin user can be canceled by the escape function. By default, this parameter is left blank, indicating that the escape function of the sysadmin and monadmin users is disabled. The value can only be **sysadmin**, **monadmin**, or an empty string. This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string, consisting of one or more characters.

Currently, this parameter supports only three values: **sysadmin**, **monadmin**, and **''**. The meanings of these values are as follows:

- **sysadmin**: Jobs of the sysadmin user can be canceled by the escape function.
- **monadmin**: Jobs of the monadmin user can be canceled by the escape function.
- **''**: The escape function of the sysadmin and monadmin users is disabled.

Default value: **''**, indicating that the escape function of the sysadmin and monadmin users is disabled.

Example:

```
resilience_escape_user_permissions = 'sysadmin,monadmin'
```


The escape function is enabled for both the sysadmin and monadmin users.

NOTICE

- You can set this parameter to multiple values separated by commas (,), for example, **resilience_escape_user_permissions = 'sysadmin,monadmin'**. You can also set this parameter to only one value, for example, **resilience_escape_user_permissions = 'monadmin'**.
- If this parameter is set for multiple times, the latest setting takes effect.
- If this parameter is set to any value in the value range, common users support the escape function.
- If a user has both the sysadmin and monadmin role permissions, the escape function of the user can be triggered only when **resilience_escape_user_permissions** is set to **'sysadmin,monadmin'**.

14.3.4.2 Disk Space

This section describes the disk space parameters, which are used to set limits on the disk space for storing temporary files.

sql_use_spacelimit

Parameter description: Specifies the space size for files to be flushed to disks when a single SQL statement is executed on a single database node. The managed space includes the space occupied by ordinary tables, temporary tables, and intermediate result sets to be flushed to disks.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from –1 to 2147483647. The unit is KB. **–1** indicates no limit.

Default value: –1

temp_file_limit

Parameter description: Specifies the limit on the size of a temporary file spilled to disk in a session. The temporary file can be a sort or hash temporary file, or the storage file for a held cursor. This is a session-level setting.

Parameter type: integer.

Unit: KB

NOTICE

This parameter does not apply to disk space used for temporary tables during the SQL query process.

Value range: –1 to 2147483647. The value **–1** indicates that there is no limit.

Default value: **–1**

Setting method: This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none

14.3.4.3 Kernel Resource Usage

This section describes kernel resource parameters. Whether these parameters take effect depends on OS settings.

max_files_per_process

Parameter description: Specifies the maximum number of simultaneously open files allowed by each server process. If the kernel is enforcing a proper limit, setting this parameter is not required.

However, on some platforms, such as most Berkeley Software Distribution (BSD) systems, the kernel allows individual processes to open many more files than the system can support. If the message "Too many open files" is displayed, set this parameter to a smaller value. Generally, the system must meet this requirement: Number of file descriptors ≥ Maximum number of concurrent statements x Number of database nodes x **max_files_per_process** x 3.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 25 to 2147483647.

Default value: 1024

shared_preload_libraries

Parameter description: Specifies one or more shared libraries to be preloaded at server start. If multiple libraries are to be loaded, separate their names using commas (,). Only the sysadmin user can access this parameter. For example, *$libdir*/**mylib** will cause **mylib.so** (or on some platforms, **mylib.sl**) to be preloaded before the loading of the standard library directory.

You can preinstall the GaussDB's stored procedure library using the '*$libdir*/pl*XXX*' syntax as described in the preceding text. *XXX* can only be **pgsql**, **perl**, **tcl**, or **python**.

By preloading a shared library and initializing it as required, the library startup time is avoided when the library is first used. However, the time to start each new server process may increase, even if that process never uses the library. Therefore, set this parameter only for libraries that will be used in most sessions.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

- If a specified library is not found, the GaussDB service will fail to start.
- Each GaussDB-supported library has a special mark that is checked to guarantee compatibility. Therefore, libraries that do not support GaussDB cannot be loaded in this way.

Value range: a string.

Default value: **security_plugin**

14.3.4.4 Cost-based Vacuum Delay

This feature allows administrators to reduce the I/O impact of the VACUUM and ANALYZE statements on concurrent database activities. It is often more important to prevent maintenance statements, such as VACUUM and ANALYZE, from affecting other database operations than to run them quickly. Cost-based vacuum delay provides a way for administrators to achieve this purpose.

NOTICE

Certain VACUUM operations hold critical locks and should be complete as quickly as possible. In GaussDB, cost-based vacuum delays do not take effect during such operations. To avoid uselessly long delays in such cases, the actual delay is the larger of the two calculated values:

- **vacuum_cost_delay** x **accumulated_balance**/**vacuum_cost_limit**
- **vacuum_cost_delay** x 4

Context

During the execution of the ANALYZE | ANALYSE and VACUUM statements, the system maintains an internal counter that keeps tracking the estimated cost of various I/O operations that are performed. For details about ANALYZE | ANALYSE and VACUUM, see "SQL Reference > SQL Syntax" in *Developer Guide*. When the accumulated cost reaches a limit (specified by **vacuum_cost_limit**), the thread performing the operation will sleep for a short period of time (specified by **vacuum_cost_delay**). Then, the counter resets and the operation continues.

By default, this feature is disabled. To enable this feature, set **vacuum_cost_delay** to a non-zero value.

vacuum_cost_delay

Parameter description: Specifies the length of time that the thread will sleep when **vacuum_cost_limit** has been exceeded.

Parameter type: integer.

Unit: ms

Value range: 0 to 100. The value **0** indicates that the cost-based vacuum delay is disabled, and a positive value indicates that the cost-based vacuum delay is

enabled. In many systems, the effective resolution of the sleep time is 10 milliseconds. Therefore, setting **vacuum_cost_delay** to a value that is not an integer multiple of 10 has the same effect as setting it to the next higher multiple of 10.

Default value: **0**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: This parameter is usually set to a small value, such as **10ms** or **20ms**. Adjusting vacuum's resource consumption is best done by changing other vacuum cost parameters.

vacuum_cost_page_hit

Parameter description: Specifies the estimated cost for vacuuming a buffer found in the shared buffer. It represents the cost to lock the buffer pool, look up the shared hash table, and scan the content of the page.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 10000.

Default value: **1**

vacuum_cost_page_miss

Parameter description: Specifies the estimated cost for vacuuming a buffer read from the disk. It represents the cost to lock the buffer pool, look up the shared hash table, read the desired block from the disk, and scan the block.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 10000.

Default value: **10**

vacuum_cost_page_dirty

Parameter description: Specifies the estimated cost charged when vacuum modifies a block that was previously clean. It represents the extra cost required to update the dirty block out to the disk again.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 10000

Default value: 20

vacuum_cost_limit

Parameter description: Specifies the cost limit. The vacuum thread will hibernate if this limit is exceeded.

Parameter type: integer.

Unit: none

Value range: 1 to 10000

Default value: **1000**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. A larger value indicates a smaller I/O frequency limit of the VACUUM, a lower probability of falling into hibernation, more efficient VACUUM, and a greater impact on service I/Os.

14.3.4.5 Background Writer

This section describes background writer parameters. The background writer thread is used to write dirty data (new or modified data) in shared buffers to disks. This mechanism ensures that database processes seldom or never need to wait for a write action to occur when handling user queries.

It also mitigates performance deterioration caused by checkpoints because the background writer continues to write dirty pages to disks and only several pages need to be written to disks at each checkpoint. This mechanism, however, increases the overall net I/O load because while a repeatedly-dirtied page may otherwise be written only once per checkpoint interval, the background writer may write it several times as it is dirtied in the same interval. In most cases, continuous light loads are preferred, instead of periodic load peaks. The parameters discussed in this section can be set based on actual requirements.

bgwriter_delay

Parameter description: Specifies the interval at which the background writer writes dirty shared buffers. Each time, the background writer initiates write operations for some dirty buffers. In full checkpoint mode, the **bgwriter_lru_maxpages** parameter is used to control the amount of data to be written each time, and the process is restarted after the period of hibernation specified by **bgwriter_delay** (in ms). In incremental checkpoint mode, the number of target idle buffer pages is calculated based on the value of **candidate_buf_percent_target**. If the number of idle buffer pages is insufficient, a batch of pages are flushed to disks at the interval specified by **bgwriter_delay** (in ms). The number of flushed pages is calculated based on the target difference percentage. The maximum number of flushed pages is limited by **max_io_capacity**.

In many systems, the effective resolution of the sleep time is 10 milliseconds. Therefore, setting this parameter to a value that is not a multiple of 10 has the same effect as setting it to a value multiple of 10.

Parameter type: integer.

Unit: millisecond

Value range: 10 to 10000

Default value: **2s** (that is, 2000 ms)

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion:

Incremental checkpoint mode: If the data volume is multiple times the value of **shared_buffers**, the value of **bgwriter_delay** cannot be greater than 2s. If the data volume is smaller than the value of **shared_buffers**, the value of **bgwriter_delay** can be increased to save I/O resources.

Full checkpoint mode: Retain the default value. If the disk capability is poor, increase the value.

candidate_buf_percent_target

Parameter description: Specifies the expected percentage of available buffers in the candidate buffer chain to **shared_buffer** when incremental checkpoints are enabled. If the actual percentage is less than the value of this parameter, the background writer thread starts to flush dirty pages that meet the conditions to disks.

Parameter type: double-precision floating point

Unit: none

Value range: 0.1 to 0.85

Default value: **0.3**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: This parameter directly affects the number of pages flushed by the background writer thread. If the data volume is greater than the value of **shared_buffers** and the parameter is set to an excessively small value, the performance is affected. If the data volume is twice or more of the value of **shared_buffers**, the value of this parameter cannot be smaller than the default value. In other scenarios, to reduce I/Os, set this parameter to a smaller value.

bgwriter_lru_maxpages

Parameter description: Specifies the number of dirty buffers the background writer can write in each round.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 1000

NOTE

When this parameter is set to **0**, the background writer is disabled. This setting does not affect checkpoints.

Default value: **100**

bgwriter_lru_multiplier

Parameter description: Specifies the coefficient used to estimate the number of dirty buffers the background writer can write in the next round.

The number of dirty buffers written in each round depends on the number of buffers used by server processes during recent rounds. The estimated number of buffers required in the next round is calculated using the following formula:

Average number of recently used buffers x **bgwriter_lru_multiplier**. The background writer writes dirty buffers until sufficient, clean, and reusable buffers are available. The number of buffers the background writer writes in each round is always equal to or less than the value of **bgwriter_lru_maxpages**.

Therefore, the value **1.0** of **bgwriter_lru_multiplier** represents a just-in-time policy of writing exactly the number of dirty buffers predicted to be required. Larger values provide some cushion against spikes in demand, whereas smaller values intentionally leave more writes to be done by server processes.

Smaller values of **bgwriter_lru_maxpages** and **bgwriter_lru_multiplier** reduce the extra I/O load caused by the background writer, but make it more likely that server processes will have to issue writes for themselves, delaying response to queries.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a floating-point number ranging from 0 to 10.

Default value: **2**

pagewriter_thread_num

Parameter description: Specifies the number of threads for background page flushing after the incremental checkpoint is enabled. Dirty pages are flushed in sequence to disks, updating recovery points.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to 16

Default value: **4**

dirty_page_percent_max

Parameter description: Specifies the percentage of dirty pages to **shared_buffers** after the incremental checkpoint is enabled. When the value of this parameter is reached, the background page flush thread flushes dirty pages based on the maximum value of **max_io_capacity**.

Parameter type: floating point

Unit: none

Value range: 0.1 to 1

Default value: **0.9**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

pagewriter_sleep

Parameter description: Specifies the interval for the page writer thread to flush dirty pages to disks after the incremental checkpoint is enabled. When the ratio of dirty pages to **shared_buffers** reaches **dirty_page_percent_max**, the number of

pages in each batch is calculated based on the value of **max_io_capacity**. In other cases, the number of pages in each batch decreases proportionally.

Parameter type: integer.

Unit: millisecond

Value range: 0 to 3600000

Default value: **2000ms** (2s)

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Do not set this parameter to a value greater than 2s. If dirty pages are generated quickly, you are advised to set this parameter to a value ranging from 100 ms to 500 ms. If this parameter is set to a large value, redo points are updated slowly, affecting Xlog recycling.

max_io_capacity

Parameter description: Specifies the maximum I/O per second for the background writer to flush pages in batches.

Parameter type: integer.

Unit: kB

Value range: 30720 to 10485760

Default value: **512000kB** (that is, 500 MB)

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**. If the value of the parameter does not contain a unit, the default unit is kB. The unit can also be MB or GB.

Setting suggestion: Set this parameter based on the service scenario and disk I/O capability of the host. If the RTO is short or the data volume is many times larger than the shared memory, and the service access data volume is random, the value of this parameter cannot be too small. A small parameter value reduces the number of pages flushed by the background writer. If a large number of pages are eliminated due to service triggering, the services are affected.

enable_consider_usecount

Parameter description: Specifies whether the backend thread considers the page popularity during page replacement. You are advised to enable this parameter in large-capacity scenarios.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**/**true**: The page popularity is considered.
- **off**/**false**: The page popularity is not considered.

Default value: **off**

dw_file_num

Parameter description: Specifies the number of doublewrite files to be written in batches. The value is related to **pagewriter_thread_num** and cannot be greater than that of **pagewriter_thread_num**. If the value is too large, it will be corrected to the value of **pagewriter_thread_num**.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to 16

Default value: **1**

dw_file_size

Parameter description: Specifies the size of each doublewrite file.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 32 to 256.

Default value: **256**

14.3.4.6 Asynchronous I/O

checkpoint_flush_after

Parameter description: Specifies the threshold for the number of pages flushed by the checkpointer thread. If the threshold is exceeded, the OS is instructed to flush the pages cached in the OS asynchronously. In GaussDB, the disk page size is 8 KB.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 256. **0** indicates that the asynchronous flush function is disabled. The size of a single page is 8 KB. You can set the number of pages or the size in bytes, for example, **32** or **256KB**. After the checkpointer thread continuously writes 32 disk pages, that is, 32 x 8 = 256 KB disk space, the asynchronous flush is performed.

Default value: **256KB**

bgwriter_flush_after

Parameter description: Specifies the threshold for the number of pages flushed by the background writer thread. If the threshold is exceeded, the background writer thread instructs the OS to asynchronously flush the pages cached in the OS to disks. In GaussDB, the disk page size is 8 KB.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 256. **0** indicates that the asynchronous flush function is disabled. The size of a single page is 8 KB. You can set the number of pages or the size in bytes, for example, **64** or **512KB**. After the

background writer thread continuously writes 64 disk pages, that is, 64 x 8 = 512 KB disk space, the asynchronous flush is performed.

Default value: **512KB** (that is, 64 pages)

backend_flush_after

Parameter description: Specifies the threshold for the number of pages flushed by the backend thread. If the number of pages exceeds the threshold, the backend thread instructs the OS to asynchronously flush the pages cached in the OS to disks. In GaussDB, the disk page size is 8 KB.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 256. **0** indicates that the asynchronous flush function is disabled. The size of a single page is 8 KB. You can set the number of pages or the size in bytes, for example, **64** or **512KB**. After the backend thread continuously writes 64 disk pages, that is, 64 x 8 = 512 KB disk space, the asynchronous flush is performed.

Default value: **0**

enable_adio_function

Parameter description: Specifies whether to enable the ADIO mode. For details about the ADIO mode, see "High Performance > ADIO and Doublewrite Removal" in *Feature Description*.

Parameter type: Boolean

Unit: none

Value range:

- **on** indicates that the ADIO mode is enabled.
- **off** indicates that the ADIO mode is disabled.

Default value: **off**

Setting method: This is a SIGHUP parameter. You can change the value of this parameter from **off** to **on** by referring to "Method 2" in **Table 14-2**. After the setting is complete, you can use the system function gs_get_io_type() to check whether the ADIO mode is enabled. For details, see "SQL Reference > Functions and Operators > System Management Functions > Other Functions" in *Developer Guide*. If you want to change the value of this parameter from **on** to **off**, restart the instance. For details, see "Method 1" in **Table 14-2**.

Setting suggestion: If the **candidate_slots** column in the DBE_PERF.global_candidate_status system view is less than 15% of the buffer page for multiple times in a short period of time, you are advised to set this parameter to **on**.

14.3.5 Data Import and Export

This section describes the parameters for importing and exporting data.

raise_errors_if_no_files

Parameter description: Specifies whether to distinguish between the problems "the number of imported file records is empty" and "the imported file does not exist." If this parameter is set to **TRUE** and the problem "the imported file does not exist" occurs, GaussDB will report the error message "file does not exist."

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **TRUE** indicates that the messages of "the number of imported file records is empty" and "the imported file does not exist" are distinguished when files are imported.
- **FALSE** indicates that the messages of "the number of imported file records is empty" and "the imported file does not exist" are not distinguished when files are imported.

Default value: **FALSE**

safe_data_path

Parameter description: Specifies the path prefix restriction except for the initial user. Currently, the path prefix restriction applies to the COPY operation and advanced packages.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string of less than 4096 characters

Default value: **NULL**

CAUTION

- If a soft link file exists in the **safe_data_path** directory, the system processes the file based on the actual file path to which the soft link points. If the actual path is not in the **safe_data_path** directory, an error is reported.
- If a hard link file exists in the **safe_data_path** directory, it can be used properly. For security purposes, exercise caution when using hard link files. Do not create hard link files that point to other directories in the **safe_data_path** directory. Ensure that the permission on the **safe_data_path** directory is minimized.

enable_copy_server_files

Parameter description: Specifies whether to enable the permission to copy server files.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the permission to copy server files is enabled.
- **off** indicates that the permission to copy server files is disabled.

Default value: **off**

NOTICE

When the **enable_copy_server_files** parameter is disabled, only the initial user is allowed to run the **COPY FROM FILENAME** or **COPY TO FILENAME** statement. When the **enable_copy_server_files** parameter is enabled, users with the SYSADMIN permission or users who inherit the gs_role_copy_files permission of the built-in role are allowed to run the **COPY FROM FILENAME** or **COPY TO FILENAME** statement.

a_format_load_with_constraints_violation

Parameter description: Specifies whether to enable **gs_loader** to support no rollback in the case of constraint conflicts.

Parameter type: string

Unit: none

Value range:

- **s1** indicates that **gs_loader** is enabled to support no rollback in the case of constraint conflicts.
- **''** indicates that **gs_loader** is not enabled to support no rollback in the case of constraint conflicts.

Default value: ''

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none.

support_binary_copy_version

Parameter description: Specifies whether the encoding information of the current database server is included when data is exported in BINARY mode using COPY FROM.

Parameter type: string

Unit: none

Value range: **''** and **header_encoding**.

Default value: **header_encoding**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. If forward compatibility is required, leave this parameter empty.

Table 14-5 Compatibility configuration items

Configuration Item	Behavior
header_encoding	When the binary mode of COPY FROM is used for export, the binary file header contains the encoding information of the current database server.
Empty	Forward compatibility configuration is performed and data is exported in the original binary format.

copy_special_character_version

Parameter description: Determines the processing of invalid characters during data import and export using COPY.

Parameter type: string

Unit: none

Value range: **''**, **no_error**, and **per_byte**.

Default value: ''

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Use gsql to connect to the database. If you use the set method, the value is case-insensitive. If you use gs_guc, the value can only be lowercase.

Setting suggestion: none

Table 14-6 Compatibility configuration items

Configuration Item	Behavior
no_error	When COPY is used to import a data file with the same encoding as that on the server, fault tolerance is performed on the data that does not meet the encoding requirements in the data file. No error is reported and the data with the original codes is inserted into the table.
per_byte	Determines how to process files encoded in GBK or ZHS16GBK when COPY is used to export text files. After the parameter is set to **per_byte**, one byte of data is exported at a time. Otherwise, two bytes of data are exported at a time. (One character occupies two bytes if data is encoded in GBK.)
Empty	The default value, which does not affect any function. Forward compatibility is supported. That is, an error is reported when invalid characters are found during COPY.

NOTICE

- To ensure that the data to be imported is valid, its encoding must be validated when it is being copied. If this parameter is enabled, verification against invalid encoding will be masked, which causes invalid characters in the field. Therefore, exercise caution before enabling this parameter.
- Currently, encoding verification is masked only when the server encoding is the same as the data encoding. Client encoding is used by default if not specified.
- To record fields where invalid encoding exists, you are advised to use the **log errors** or **log errors data** parameter in the COPY syntax.
- In binary mode, **copy_special_character_version** is set to **'no_error'**, and it takes effect only for fields of the TEXT, CHAR, VARCHAR, NVARCHAR2, or CLOB type.
- This parameter is valid only in the database with character sets encoded in UTF-8, GB18030, GB18030_2022, ZHS16GBK, or LATIN1.
- When the encoding of both the client and server is GBK or ZHS16GBK and the database contains data encoded in an invalid format, if **copy_special_character_version** is not set to **per_byte**, the exported data file may contain unexpected data.
- If **copy_special_character_version** is set to **no_error**, this parameter cannot be used together with the **COMPATIABLE_ILLEGAL_CHARS** parameter in COPY.

14.3.6 Write Ahead Log

14.3.6.1 Settings

wal_level

Parameter description: Specifies the level of information to be written to the WAL. The value cannot be empty or commented out.

NOTICE

- To enable WAL archiving and data streaming replication between the primary and standby nodes, set this parameter to **archive**, **hot_standby**, or **logical**.
- If this parameter is set to **minimal**, **hot_standby** must be set to **off**, and **max_wal_senders** must be set to **0**. In addition, the database must be deployed in a standalone system. Otherwise, the database cannot be started.
- If this parameter is set to **archive**, **hot_standby** must be set to **off**. Otherwise, the database startup fails. However, **hot_standby** cannot be set to **off** in an HA cluster. For details, see the description of the **hot_standby** parameter.

Value type: enumerated type

Value range:

- **minimal**

Advantages: Certain bulk operations (including creating tables and indexes, executing cluster operations, and copying tables) are safely skipped in logging, which can make those operations much faster.

Disadvantages: WALs contain only basic information required for recovery from a database server crash or an emergency shutdown. Data cannot be restored from archived WALs.

- **archive**

  Adds logging required for WAL archiving, supporting the database restoration from archives.

- **hot_standby**
  - Further adds information required to run SQL queries on a standby node and takes effect after the database restarts.
  - To enable read-only queries on a standby node, the **wal_level** parameter must be set to **hot_standby** on the primary node and the same value must be set on the standby node. There are few measurable differences in performance between using **hot_standby** and **archive** levels. However, feedback is welcome if any differences in their impacts on product performance are noticeable.

- **logical**

  This parameter indicates that WALs support logical replication.

Default value: **hot_standby**

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Set this parameter to **logical** when the logical replication function is enabled. In other scenarios, set this parameter to **hot_standby**.

fsync

Parameter description: Specifies whether the GaussDB server uses the fsync() function (see **wal_sync_method**) to ensure that updates can be written to disks in a timely manner.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

- Using the fsync() function ensures that the data can be recovered to a known state when the OS or hardware crashes.
- Setting this parameter to **off** may result in unrecoverable data corruption in a system crash.

Value range: Boolean

- **on** indicates that the fsync() function is used.
- **off** indicates that the fsync() function is not used.

Default value: **on**

synchronous_commit

Parameter description: Specifies the synchronization mode of the current transaction.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Generally, logs generated by a transaction are synchronized in the following sequence:

1. The primary node writes the logs to the local memory.
2. The primary node writes the logs in the local memory to the local file system.
3. The primary node flushes the logs in the local file system to disks.
4. The primary node sends the logs to the standby node.
5. The standby node receives the logs and saves them to the local memory.
6. The standby node writes the logs in the local memory to the local file system.
7. The standby node flushes the logs in the local file system to disks.
8. The standby node replays the logs to complete the incremental update of data files.

Value range: enumerated values

- **on**: The primary node waits for the standby node to flush logs to disks before committing a transaction.
- **off**: The primary node commits transactions without waiting for the primary node to flush logs to disks. This mode is also called asynchronous commit.
- **local**: The primary node waits for the primary node to flush logs to disks before committing a transaction. This mode is also called local commit.
- **remote_write**: The primary node waits for the standby node to write logs to the file system before committing a transaction. (The logs do not need to be flushed to disks.)
- **remote_receive**: The primary node waits for the standby node to receive logs before committing a transaction. (The logs do not need to be written to the file system.)
- **remote_apply**: The primary node waits for the standby node to complete log replay before committing a transaction.
- **true**: same as **on**.
- **false**: same as **off**.
- **yes**: same as **on**.
- **no**: same as **off**.
- **1**: same as **on**.
- **0**: same as **off**.
- **2**: same as **remote_apply**.

Default value: **on**

NOTICE

This parameter is maintained by the CM. If it is manually modified, data may be lost. For details, see "Unified Cluster Management Tool > Features > Automatic Copy Addition and Reduction by Shard > Remarks" in *Tool Reference*.

wal_sync_method

Parameter description: Specifies the method used for forcing WAL updates out to disk.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

If **fsync** is set to **off**, the setting of this parameter does not take effect because WAL updates will not be forcibly written to disks.

Value range: enumerated values

- **open_datasync** indicates that WAL files are opened with the **O_DSYNC** option.
- **fdatasync** indicates that fdatasync() is called at each commit (SUSE 10 and SUSE 11 are supported).
- **fsync_writethrough** indicates that fsync() is called at each commit to force data in the buffer to be written to disks.

  NOTE

  wal_sync_method can be set to **fsync_writethrough** on a Windows platform, but this setting has the same effect as setting the parameter to **fsync** on the Windows platform.
- **fsync** indicates that fsync() is called at each commit. (SUSE 10 and SUSE 11 are supported.)
- **open_sync** indicates that open() with the **O_SYNC** option is used to write WAL files (SUSE 10 and SUSE 11 are supported).

  NOTE

  Not all platforms support the preceding parameters.

Default value: fdatasync

full_page_writes

Parameter description: Specifies whether the GaussDB server writes the entire content of each disk page to WALs during the first modification of that page after a checkpoint. When both **enable_incremental_checkpoint** and **enable_double_write** are enabled, **full_page_writes** is not used.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

- This parameter is needed because a page write that is in process during an OS crash might be only partially completed, leading to an on-disk page that contains a mix of old and new data. The row-level change data normally stored in WALs will not be enough to completely restore such a page during post-crash recovery. Storing the full page image guarantees that the page can be correctly restored, but at the price of increasing the amount of data that must be written to WALs.
- Setting this parameter to **off** might lead to unrecoverable data corruption after a system failure. It might be safe to set this parameter to **off** if you have hardware (such as a battery-backed disk controller) or file-system software (such as ReiserFS 4) that reduces the risk of partial page writes to an acceptably low level.

Value range: Boolean

- **on** indicates that this feature is enabled.
- **off** indicates that this feature is disabled.

Default value: **on**

wal_log_hints

Parameter description: Specifies whether to write an entire page to WALs during the first modification of that page after a checkpoint, even for non-critical modifications of so-called hint bits. You are advised not to modify the setting.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the entire page is written to WALs.
- **off** indicates that the entire page is not written to WALs.

Default value: **on**

wal_buffers

Parameter description: Specifies the size of shared memory for storing WALs, in multiples of **XLOG_BLCKSZ** or the actual size. **XLOG_BLCKSZ** indicates the size of an Xlog block, typically 8 KB.

Parameter type: integer.

Unit: 8 kB (**XLOG_BLCKSZ** size)

Value range: **–1**, or 4 to 2^{18}. The minimum value is **–1** and the maximum value is **262144** (number of **XLOG_BLCKSZ**).

- If this parameter is set to **–1**, the value of **wal_buffers** is automatically adjusted based on the value of **shared_buffers**. The default value of **wal_buffers** is 1/32 of **shared_buffers**. The minimum value is **XLOG_BLCKSZ** multiplied by 8 and the maximum value is **XLOG_BLCKSZ** multiplied by 2048.

If the automatically adjusted value is less than the minimum value, the value is forcibly set to the minimum value. If the value is greater than the maximum value, the value is forcibly set to the maximum value.

- If this parameter is set to a value smaller than 4 (that is, **0**, **1**, **2**, or **3**), the value **4** is forcibly used.

Default value:

1GB (196-core CPU/1536 GB memory, 128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 64-core CPU/512 GB memory, 60-core CPU/480 GB memory, 32-core CPU/256 GB memory); **512MB** (16-core CPU/128 GB memory); **256MB** (8-core CPU/64 GB memory); **128MB** (4-core CPU/32 GB memory); **16MB** (4-core CPU/16 GB memory)

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**. For example, the value **2048** indicates that **wal_buffers** is 2048 x 8 KB, and the value **20480kB** indicates that **wal_buffers** is **20480KB**. If the value contains a unit, the value must be KB, MB, or GB and must be an integer multiple of 8 KB.

Setting suggestion: The content of the WAL buffer is written to disks at every transaction commit. Therefore, setting an extremely large value is unlikely to bring a significant increase in system performance. However, setting this parameter to a few megabytes can improve the disk write performance on a server where a large number of transactions are committed at the same time. The default value meets user requirements in most cases. You are advised to retain it. Do not set **wal_buffers** to an excessively large or small value, and the following condition must be true: **data_replicate_buffer_size** + **segment_buffers** + **shared_buffers** + **wal_buffers** + **temp_buffers** + **maintenance_work_mem** + **work_mem** + **query_mem** + (Standby node) **wal_receiver_buffer_size** < **max_process_memory** < Memory size of the physical machine. If the value of the memory parameter is too large and exceeds the upper limit of the physical memory, the database fails to be started because no sufficient memory can be allocated.

wal_writer_delay

Parameter description: Specifies the delay between activity rounds for the WAL writer.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

A longer delay might lead to insufficient WAL buffer and a shorter delay leads to continuously writing of the WALs, thereby increasing the load of disk I/O.

Value range: an integer ranging from 1 to 10000. The unit is millisecond.

Default value: **200ms**

commit_delay

Parameter description: Specifies the duration for committed data to be stored in the WAL buffer.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

- When this parameter is set to a non-zero value, the committed transaction is stored in the WAL buffer instead of being written to the WAL immediately. Then the WAL writer flushes the buffer to disks periodically.
- If system load is high, other transactions are probably ready to be committed within the delay. If no other transactions are ready to be committed, the delay is a waste of time.

Value range: an integer ranging from 0 to 100000. The unit is μs. **0** indicates no delay.

Default value: **0**

commit_siblings

Parameter description: Specifies a threshold on the number of concurrent transactions when a transaction initiates a commit request. If the number of concurrent transactions is greater than the value of this parameter, the transaction will wait for a period of time specified by **commit_delay**. Otherwise, this transaction is written into WALs immediately.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 1000

Default value: **5**

wal_block_size

Parameter description: Specifies the size of a WAL disk block.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

Value range: an integer. The unit is byte.

Default value: **8192**

wal_segment_size

Parameter description: Specifies the size of a WAL segment file.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

Value range: an integer. The unit is 8 KB.

Default value: **16MB** (2048 x 8 KB)

walwriter_cpu_bind

Parameter description: Binds the WAL writer thread to a specified CPU core.

Parameter type: integer.

Unit: none

Value range: –1 to the number of cores minus 1. CPU cores are numbered starting from 0.

- **–1**: indicates that CPU core binding is not performed.
- *N*: indicates that the WAL writer thread is bound to CPU core *N*.

Default value: **–1**

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

wal_sender_bind_cpu_attr

Parameter description: Specifies the core binding operation of the log sender thread. Only the sysadmin user can access this parameter.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

- **'nobind'**: The thread is not bound to a core.
- **'cpuorderbind: 8-12'**: One thread is bound to one core starting from core 8. If the number of cores in the range is insufficient, the remaining threads are not bound. You are advised to set the interval to a value greater than or equal to the value of **max_wal_senders**.

Value range: a string of more than 0 characters. The value is case-insensitive.

Default value: **'nobind'**

walwriteraux_bind_cpu

Parameter description: Specifies the CPU core bound to the auxiliary log writer thread.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from –1 to the number of cores minus 1.

Default value: **–1**

walwriter_sleep_threshold

Parameter description: Specifies the number of times that idle Xlogs are refreshed before the Xlog flusher enters the sleep state. If the number of times reaches the threshold, the Xlog flusher enters the sleep state.

Parameter type: integer.

Unit: none

Value range: 1 to 50000

Default value: **500** (196-core CPU/1536 GB memory, 128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 64-core CPU/512 GB memory, and 60-core CPU/480 GB memory); **50** (32-core CPU/256 GB memory, 16-core CPU/128 GB memory, 8-core CPU/64 GB memory, 4-core CPU/32 GB memory, and 4-core CPU/16 GB memory)

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

wal_file_init_num

Parameter description: Specifies the number of Xlog segment files created at a time when the WAL writer thread is started.

Parameter type: integer.

Unit: none

Value range: 0 to 1000000

Default value: **10** (196-core CPU/1536 GB memory, 128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 64-core CPU/512 GB memory, and 60-core CPU/480 GB memory); **0** (32-core CPU/256 GB memory, 16-core CPU/128 GB memory, 8-core CPU/64 GB memory, 4-core CPU/32 GB memory, and 4-core CPU/16 GB memory)

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

wal_file_preinit_bounds

Parameter description: Specifies the maximum number of WAL segment files that can be pre-expanded by the WAL writer auxiliary thread per second during service running. The WAL segment file size is 16 MB. If this parameter is set to **0**, there is no restriction.

Parameter type: integer

Unit: none

Value range: [0,1024]

Default value: **0**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. Set this parameter to a valid value based on the disk I/O capability. For details, see the value of **max_io_capacity**.

- If **max_io_capacity** is set to **500MB**, set this parameter to **25**.
- If **max_io_capacity** is set to **1GB**, set this parameter to **50**.

xlog_file_path

Parameter description: Specifies the path of the Xlog shared disk in dual-database instance shared storage scenarios. This parameter is configured by the OM during database system initialization. You are advised not to modify the configuration.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string.

Default value: **NULL**

xlog_file_size

Parameter description: Specifies the size of the Xlog shared disk in dual-database instance shared storage scenarios. This parameter is configured by the OM during database system initialization. You are advised not to modify the configuration.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a long integer ranging from 5053733504 to 576460752303423487. The unit is byte.

Default value: **549755813888**

xlog_lock_file_path

Parameter description: Specifies the path of the lock file preempted by the Xlog shared disk in dual-database instance shared storage scenarios. This parameter is configured by the OM during database system initialization. You are advised not to modify the configuration.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string.

Default value: **NULL**

max_size_for_shared_storage_xlog_write

Parameter description: Specifies the maximum amount of data that can be written to a shared disk at a time.

Parameter type: integer.

Unit: KB

Value range: 8–131072

Default value: 1024

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none

force_promote

Parameter description: Specifies whether to enable the forcible switchover function on the standby node.

When a database instance is faulty, the forcible switchover enables the database instance to recover services as soon as possible at the cost of losing some data. This is an escape method used when the database instance is unavailable. You are advised not to trigger this method frequently. You are advised not to use this function if you are not clear about the impact of data loss on services.

To use this function, you need to enable it on the DN and CM Server and restart the database instance for the setting to take effect. For details about how to enable the forcible switchover function on the standby node, see "Emergency Handling > Performing a Forcible Primary/Standby Switchover" in *Troubleshooting*.

Value range: **0** or **1**

The value **0** indicates that the function is disabled, and the value **1** indicates that the function is enabled.

Default value: **0**

wal_debug

Parameter description: Specifies whether to output WAL-related debugging information. This parameter is available only when **WAL_DEBUG** is enabled during compilation.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

Default value: **false**

wal_flush_timeout

Parameter description: Specifies the timeout interval for traversing **WalInsertStatusEntryTbl**. It is the maximum wait time for the adaptive Xlog disk flushing I/O to traverse **WalInsertStatusEntryTbl**.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

If the timeout interval is too long, the Xlog flushing frequency may decrease, reducing the Xlog processing performance.

Value range: an integer ranging from 0 to 90000000 (μs)

Default value:

2us (196-core CPU/1536 GB memory, 128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 64-core CPU/512 GB memory, 60-core CPU/480 GB memory, 32-core CPU/256 GB memory, 16-core CPU/128 GB memory, 8-core CPU/64 GB memory, 4-core CPU/32 GB memory); **800us** (4-core CPU/16 GB memory)

wal_flush_delay

Parameter description: Specifies the wait interval when an entry in the **WAL_NOT_COPIED** state is encountered during **WalInsertStatusEntryTbl** traversal.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 90000000 (μs)

Default value:

1us (196-core CPU/1536 GB memory, 128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 64-core CPU/512 GB memory, 60-core CPU/480 GB memory, 32-core CPU/256 GB memory, 16-core CPU/128 GB memory, 8-core CPU/64 GB memory, 4-core CPU/32 GB memory); **100us** (4-core CPU/16 GB memory)

wal_receiver_bind_cpu

Parameter description: Specifies the CPU core ID bound to the WAL receiver thread.

The default value is **–1**, which is an invalid value, indicating that cores are not bound.

Although the value range is *INT_MAX*, the actual value is related to the number of used device cores. If the specified CPU core ID does not exist, an error is reported and core binding is interrupted.

Multiple threads can be bound to one core, but the performance deteriorates. Therefore, you are advised not to bind multiple threads to one core.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a value ranging from –1 to *INT_MAX*.

Default value: **–1**

wal_rec_writer_bind_cpu

Parameter description: Specifies the CPU core ID bound to the WAL rec writer thread.

The default value is **–1**, which is an invalid value, indicating that cores are not bound.

Although the value range is *INT_MAX*, the actual value is related to the number of used device cores. If the specified CPU core ID does not exist, an error is reported and core binding is interrupted.

Multiple threads can be bound to one core, but the performance deteriorates. Therefore, you are advised not to bind multiple threads to one core.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a value ranging from –1 to *INT_MAX*.

Default value: **–1**

14.3.6.2 Checkpoints

checkpoint_segments

Parameter description: Specifies the minimum number of WAL segment files in the period specified by **checkpoint_timeout**. The size of each log file is 16 MB.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer. The minimum value is **1**.

Increasing the value of this parameter speeds up the import of a large amount of data. Set this parameter based on **checkpoint_timeout** and **shared_buffers**. This parameter affects the number of WAL segment files that can be reused. Generally, the maximum number of reused files in the **pg_xlog** folder is twice the number of **checkpoint_segments**. The reused files are not deleted and are renamed to the WAL segment files which will be later used.

Default value: 1024

checkpoint_timeout

Parameter description: Specifies the maximum time between automatic WAL checkpoints.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 30 to 3600. The unit is s.

If the value of **checkpoint_segments** is increased, you need to increase the value of this parameter. The increase of these two parameters further requires the increase of **shared_buffers**. Consider all these parameters during setting.

Default value: **15min**

checkpoint_completion_target

Parameter description: Specifies the target of checkpoint completion.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a double-precision floating-point number ranging from 0.0 to 1.0

Default value: **0.5**

NOTE

The default value **0.5** indicates that each checkpoint should be complete within 50% of the interval between checkpoints.

checkpoint_warning

Parameter description: Specifies a time in seconds. If the checkpoint interval is close to this time due to filling of checkpoint segment files, a message is sent to the server log to suggest an increase in the **checkpoint_segments** value.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to *INT_MAX*. The unit is s. **0** indicates that the warning is disabled.

Default value: **5min**

Recommended value: **5min**

checkpoint_wait_timeout

Parameter description: Sets the longest time that the checkpoint waits for the checkpointer thread to start.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 2 to 3600. The unit is s.

Default value: **1min**

enable_incremental_checkpoint

Parameter description: Specifies whether to enable incremental checkpoint.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

Default value: **on**

enable_double_write

Parameter description: Specifies whether to enable double writing. When the incremental checkpoint function is enabled and **enable_double_write** is enabled, the **enable_double_write** dual-write feature is used for protection and **full_page_writes** is not used to prevent half-page write.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

Default value: **on**

incremental_checkpoint_timeout

Parameter description: Specifies the maximum interval between automatic WAL checkpoints when the incremental checkpoint is enabled.

Parameter type: integer

Unit: second

Value range: 1 to 3600

Default value: **60**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

enable_xlog_prune

Parameter description: Specifies whether the primary node recycles logs if the size of Xlogs exceeds the value of **max_size_for_xlog_prune** when any standby node is disconnected.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- If this parameter is set to **on**, the primary node recycles logs when any standby node is disconnected.
- If this parameter is set to **off**, the primary node does not recycle logs when any standby node is disconnected.

Default value: **on**

max_size_for_xlog_prune

Parameter description: Specifies the maximum number of Xlogs retained on the primary node when the standby node is faulty. This parameter takes effect when **enable_xlog_prune** is enabled. The mechanism is as follows:

1. If all standby nodes specified by the replconninfo series GUC parameters are connected to the primary node, this parameter does not take effect.
2. This parameter takes effect when one or more standby nodes specified by the replconninfo series GUC parameters are disconnected to the primary node. If the number of Xlogs on the host is greater than the value of this parameter, the Xlogs will be forcibly recycled. Exception: In synchronous commit mode (that is, the **synchronous_commit** parameter is not set to **local** or **off**), if there are standby nodes connecting to the primary node, the primary node retains the logs after the minimum log receiving location in the majority of standby nodes. In this case, the number of retained logs may be greater than the value of **max_size_for_xlog_prune**.
3. If a standby node is being built, this parameter does not take effect. All logs on the primary node are retained to prevent failures caused by log recycling during the build operation.

Parameter type: integer

Unit: KB

Value range: 0 to 2147483647

Default value: **256GB**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**. If the value of the parameter does not contain a unit, the default unit is KB. The unit can also be MB or GB.

Setting suggestion: If the disk space is small, you are advised to set this parameter to a small value. The maximum value is **256GB**.

14.3.6.3 Log Replay

recovery_time_target

Parameter description: Specifies the time for a standby node to write and replay logs.

Parameter type: integer

Unit: second

Value range: 0 to 3600

0 indicates that log flow control is disabled. A value from **1** to **3600** indicates that a standby node can write and replay logs within the period specified by the value, so that the standby node can quickly assume the primary role. If this parameter is set to a small value, the performance of the primary node is affected. If it is set to a large value, the log flow is not effectively controlled.

Default value: **60**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

recovery_max_workers

Parameter description: Specifies the maximum number of concurrent replayer threads.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 20

Default value: **4** (For better performance, the default value in tool installation is **4**.)

queue_item_size

Parameter description: Specifies the maximum length of the task queue of each redo replayer thread.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a value ranging from 1 to 65535.

Default value: **560**

recovery_parse_workers

Parameter description: Specifies the number of **ParseRedoRecord** threads for the ultimate RTO feature.

1. In addition, it must be used together with **recovery_redo_workers**. If both **recovery_parse_workers** and **recovery_redo_workers** are greater than 1, ultimate RTO is enabled. If you do not want to enable ultimate RTO, retain the default value **1** of **recovery_parse_workers**.
2. Ensure that the value of this parameter **replication_type** is set to **1** when ultimate RTO is enabled.
3. If both the ultimate RTO and parallel replay are enabled at the same time, the ultimate RTO feature takes effect but the parallel replay feature does not take effect.
4. Ultimate RTO does not support flow control. Flow control is determined by the parameter **recovery_time_target**.

Parameter type: integer

Unit: none

Value range: 1 to 16

Default value: **1**

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: For details about the values of **recovery_parse_workers** and **recovery_redo_workers** for different CPUs, memories, and deployment modes, see **Table 1 Parameter settings for different CPUs, memory sizes, and deployment modes**.

NOTE

After ultimate RTO is enabled, the total number of extra replayer threads started by the standby node = **recovery_parse_workers** x (**recovery_redo_workers** + 2) + 5. More replayer threads occupy more CPU, memory, and I/O resources. Set parameters based on the actual hardware configuration. If the parameter value is too large, the CPU and memory usage may be too high, causing system startup exceptions.

recovery_redo_workers

Parameter description: Specifies the number of **PageRedoWorker** threads corresponding to each **ParseRedoRecord** thread when the ultimate RTO feature is enabled. **recovery_redo_workers** must be used together with **recovery_parse_workers**. The value of **recovery_redo_workers** takes effect only when **recovery_parse_workers** is greater than 1.

Parameter type: integer

Unit: none

Value range: 1 to 8

Default value: **1**

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: For details about the values of **recovery_parse_workers** and **recovery_redo_workers** for different CPUs, memories, and deployment modes, see **Table 1 Parameter settings for different CPUs, memory sizes, and deployment modes**.

Table 14-7 Parameter settings for different CPUs, memory sizes, and deployment modes

No.	Number of CPUs	Memory (GB)	Hybrid Deployment or Not	recovery_parse_workers	recovery_redo_workers	Number of Replayer Threads	Remarks
1	4	-	-	1	1	-	Ultimate RTO is not recommended.
2	8	-	Yes	1	1	-	Ultimate RTO is not recommended.
3	8	64	No	1	1	-	Ultimate RTO is not recommended.
4	16	128	Yes	1	1	-	Ultimate RTO is not recommended.
5	16	128	No	2	3	15	-
6	32	256	Yes	2	2	13	-
7	32	256	No	2	8	25	-
8	64	512	Yes	2	4	17	-

No.	Number of CPUs	Memory (GB)	Hybrid Deployment or Not	recovery_parse_workers	recovery_redo_workers	Number of Replayer Threads	Remarks
9	64	512	No	2	8	25	Set the parameter to the recommended value for larger hardware specifications.
10	96	768	-	2	8	25	Set the parameter to the recommended value for larger hardware specifications.

recovery_parallelism

Parameter description: Specifies the actual number of replayer threads. This parameter is read-only.

This is a POSTMASTER parameter and is affected by **recovery_max_workers** and **recovery_parse_workers**. If any value is greater than 0, **recover_parallelism** will be recalculated.

Value range: an integer ranging from 1 to 2147483647

Default value: **1**

enable_page_lsn_check

Parameter description: Specifies whether to enable the data page LSN check. During replay, the current LSN of the data page is checked to see if it is the expected one.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

Default value: **on**

recovery_min_apply_delay

Parameter description: Specifies the replay delay of the standby node.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

- This parameter does not take effect on the primary node. It must be set on the standby node that requires a delay. You are advised to set this parameter on the asynchronous standby node. If the delay is set on the asynchronous standby node, the RTO will take a long time after the node is promoted to primary.
- The delay time is calculated based on the transaction commit timestamp on the primary node and the current time on the standby node. Therefore, ensure that the clocks of the primary and standby nodes are synchronized.
- If the delay time is too long, the disk where the Xlog file is located on the standby node may be full. Therefore, you need to set the delay time based on the disk size.
- Operations without transactions are not delayed.
- After the primary/standby switchover, if the original primary node needs to be delayed, you need to manually set this parameter.
- When **synchronous_commit** is set to **remote_apply**, synchronous replication is affected by the delay. Each commit message is returned only after the replay on the standby node is complete.
- Using this feature also delays **hot_standby_feedback**, which may cause the primary node to bloat, so be careful when using both.
- If a DDL operation (such as DROP or TRUNCATE) that holds an AccessExclusive lock is performed on the primary node, the query operation on the operation object on the standby node will be returned only after the lock is released during the delayed replay of the record on the standby node.

Value range: an integer ranging from 0 to INT_MAX. The unit is ms.

Default value: **0** (no delay added)

dcf_truncate_dump_info_level

Parameter description: Specifies whether to print the LSN truncated by the DCF and the subsequent LSNs.

Parameter type: integer

Unit: none

Value range: 0 to 2

- **0**: disabled.
- **1**: prints all LSNs truncated by the DCF (Xlogs whose LSN is greater than or equal to the truncated LSN).
- **2**: prints all LSNs truncated by the DCF and prints warning-level logs when the LSNs flushed to disks are greater than the truncated LSNs.

Default value: **0** (disabled)

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

redo_bind_cpu_attr

Parameter description: Specifies the core binding operation of the replayer thread. Only the sysadmin user can access this parameter. This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string of more than 0 characters. The value is case-insensitive.

- **'nobind'**: The thread is not bound to a core.
- **'nodebind: 1, 2'**: Use the CPU cores in NUMA groups 1 and 2 to bind threads.
- **'cpubind: 0-30'**: Use the CPU cores 0 to 30 to bind threads.
- **'cpuorderbind: 16-32'**: One thread is bound to one CPU core starting from core 16. If the number of cores in the range is insufficient, the remaining threads are not bound. You are advised to set the interval to a value greater than or equal to the value of **recovery_parallelism** plus 1.

Default value: **'nobind'**

14.3.6.4 Archiving

archive_timeout

Parameter description: Specifies the archiving period.

This is a **SIGHUP** parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

- The server is forced to switch to a new WAL segment file when the period specified by this parameter has elapsed since the last file switch.
- Archived files that are closed early due to a forced switch are still of the same length as full files. Therefore, a very short **archive_timeout** will bloat the archive storage. You are advised to set **archive_timeout** to **60s**.

Value range: an integer ranging from 0 to 1073741823. The unit is s. **0** indicates that archiving timeout is disabled.

Default value: **0**

archive_interval

Parameter description: Specifies the archiving interval.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

- Log files are forcibly archived when the period specified by this parameter has elapsed.
- Archiving involves I/O operations. Therefore, frequent archiving is not allowed. In addition, the RPO cannot be set to a large value; otherwise, the PITR will be affected. You are advised to use the default value.

Value range: an integer ranging from 1 to 1000. The unit is second.

Default value: **1**

14.3.7 HA Replication

14.3.7.1 Sending Server

max_wal_senders

Parameter description: Specifies the maximum number of concurrent connections of the transaction log sender. The value must be smaller than that of **max_connections**.

NOTICE

wal_level must be set to **archive**, **hot_standby**, or **logical** to allow the connection from standby nodes.

Parameter type: integer.

Value range: 0 to 1024. The recommended value range is 8 to 100.

Default value: 20

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestions: 1. It can be set to **0** only when a single DN is used and there is no primary/standby instance. 2. When HA replication, backup and restoration, and logical decoding are used, you are advised to set this parameter to a value by referring to the following formula: Number of current standby nodes + Number of backup connections + Number of required logical replication connections. If the actual value is smaller than the recommended value, these functions may be unavailable or abnormal.

wal_keep_segments

Parameter description: Specifies the number of Xlog file segments. Specifies the minimum number of transaction log files stored in the **pg_xlog** directory. The standby node obtains log files from the primary node for streaming replication.

Parameter type: integer.

Value range: 2 to *INT_MAX*.

Default value: **128**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion:

- During WAL archiving or recovery from a checkpoint on the server, the system may retain more log files than the number specified by **wal_keep_segments**.
- If this parameter is set to an excessively small value, a transaction log may have been overwritten by a new transaction before requested by the standby node. As a result, the request fails and the connection between the primary and standby nodes is terminated.
- If the HA system uses asynchronous transmission, increase the value of **wal_keep_segments** when data greater than 4 GB is continuously imported in COPY mode. Take T6000 board as an example. If the data to be imported reaches 50 GB, you are advised to set this parameter to **1000**. You can dynamically restore the setting of this parameter after data import is complete and the log synchronization is normal.
- If the synchronous_commit level is lower than LOCAL_FLUSH, you are advised to set this parameter to **1000** when rebuilding the standby node to prevent rebuilding failures caused by primary node log recycling during the rebuilding.

wal_sender_timeout

Parameter description: Specifies the maximum duration that the sender waits for the receiver to receive transaction logs.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

- If the data volume on the primary node is huge, the value of this parameter must be increased for the database rebuilding on a standby node. For example, if the data volume on the primary node reaches 500 GB, you are advised to set this parameter to 600 seconds.
- This parameter cannot be set to a value larger than the value of **wal_receiver_timeout** or the timeout parameter for database rebuilding.

Value range: an integer ranging from 0 to *INT_MAX*. The unit is ms.

Default value: **6s**

max_replication_slots

Parameter description: Specifies the number of log replication slots on the primary node.

Parameter type: integer.

Value range: 0 to 1024. The recommended value range is 8 to 100.

Default value: 20

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion:

When HA replication, backup and restoration, and logical decoding are used, you are advised to set this parameter to a value by referring to the following formula: Number of current physical streaming replication slots + Number of backup slots + Number of required logical replication slots. If the actual value is smaller than the recommended value, these functions may be unavailable or abnormal.

- Physical streaming replication slots provide an automatic method to ensure that Xlogs are not removed from a primary node before they are received by all the standby nodes. That is, physical streaming replication slots are used to support primary/standby HA. The number of physical streaming replication slots required by a database is equal to the ratio of standby nodes to the primary node. For example, if an HA database has one primary node and one standby node, the number of required physical streaming replication slots will be one. If an HA database has one primary node and three standby nodes, the number of required physical streaming replication slots will be three.
- Backup slot records replication information during backup execution. Full backup and incremental backup correspond to two independent backup slots.
- Plan the number of logical replication slots as follows:
  - A logical replication slot can carry changes of only one database for decoding. If multiple databases are involved, create multiple logical replication slots.
  - If logical replication is needed by multiple target databases, create multiple logical replication slots in the source database. Each logical replication slot corresponds to one logical replication link.
  - A maximum of 20 logical replication slots can be enabled for decoding on the same instance.

enable_slot_log

Parameter description: Specifies whether to enable primary/standby synchronization for replication slots. Currently, only archive slots and backup slots are involved.

Parameter type: Boolean.

Value range:

- **on** indicates that primary/standby synchronization is enabled for replication slots.
- **off** indicates that primary/standby synchronization is disabled for replication slots.

Default value: **on**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Set this parameter to **on** only in cloud scenarios where logical replication is enabled. In other scenarios, set this parameter to **off**.

max_changes_in_memory

Parameter description: Specifies the maximum number of DML statements cached in memory for a single transaction during logical decoding.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to 2147483647.

Default value: **4096**

max_cached_tuplebufs

Parameter description: Specifies the upper limit of the total tuple information cached in the memory during logical decoding. You are advised to set this parameter to a value greater than or equal to twice of **max_changes_in_memory**.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to 2147483647.

Default value: **8192**

logical_decode_options_default

Parameter description: Specifies the global default value for unspecified decoding options when logical decoding starts.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Currently, the following logical decoding options are supported: **parallel-decode-num**, **parallel-queue-size**, **max-txn-in-memory**, **max-reorderbuffer-in-memory**, **exclude-users**, and **skip-generated-columns**. For details about the options, see "Logical Replication > Logical Decoding > Logical Decoding Options" in *Feature Guide*.

Value range: a string of key=value characters separated by commas (,), for example, **'parallel-decode-num=4,parallel-queue-size=128,exclude-users=userA,skip-generated-columns=on'**. An empty string indicates that the default value of the program is used.

Default value: **""**

NOTICE

- The SIGHUP parameter does not affect the started logical decoding process. The options specified by this parameter are used as the default settings for subsequent logical decoding startup, and the settings specified in the startup command are preferentially used.
- The **exclude-users** option is different from the logical decoding startup option. You are not allowed to specify multiple blacklisted users.

logical_sender_timeout

Parameter description: Specifies the maximum waiting time for the sender to wait for the receiver to receive logical logs.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 2147483647. The unit is ms.

Default value: **30s**

enable_logicalrepl_xlog_prune

Parameter description: Specifies whether to enable the function of forcibly invalidating a logical replication slot. When the current GUC parameters **enable_logicalrepl_xlog_prune** is set to **on**, **enable_xlog_prune** is set to **on**, and **max_size_for_xlog_retention** is set to a non-zero value, the number of reserved log segments caused by the backup slot or logical replication slot exceeds the value of **wal_keep_segments**, and other replication slots do not cause more reserved log segments, if the value of **max_size_for_xlog_retention** is greater than 0 and the number of retained log segments (the size of each log segment is 16 MB) caused by the current logical replication slot exceeds the value of **max_size_for_xlog_retention**, or if the value of **max_size_for_xlog_retention** is less than 0 and the disk usage reaches the value of **–max_size_for_xlog_retention**/**100**, the logical replication slot is forcibly invalidated and **restart_lsn** is set to **7FFFFFFF/FFFFFFFF**.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **true** indicates that the function of forcibly invalidating a logical replication slot is enabled.
- **false** indicates that the function of forcibly invalidating a logical replication slot is disabled.

Default value: **false**

enable_logical_replication_ddl

Parameter description: Specifies whether the logical decoding supports DDL statements, reverse parsing, and log generation.

Parameter type: Boolean.

Value range:

- **on**: Logical replication supports DDL statements, reversely parses DDL execution results, and generates DDL WALs.
- **off**: DDL statements are not supported, reverse parsing is not performed, and WALs are not generated.

Default value: **on**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

enable_wal_shipping_compression

Parameter description: Specifies whether to enable cross-database instance log compression in streaming DR mode.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

This parameter applies only to a pair of WAL sender and WAL receiver for cross-database instance transmission in streaming DR and is configured on the primary database instance.

Value range: Boolean

- **true** indicates that cross-database instance log compression is enabled for streaming DR.
- **false** indicates that cross-database instance log compression is disabled for streaming DR.

Default value: **false**

repl_auth_mode

Parameter description: Specifies the validation mode for primary/standby replication and standby node rebuilding.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

- If UUID validation is enabled on the primary node and a non-null repl_uuid validation code is configured, UUID validation must also be enabled on the standby node and the same repl_uuid validation code must be configured on the standby node. Otherwise, requests for log replication between the primary and standby nodes and standby node rebuilding will be rejected by the primary node.
- The SIGHUP parameter can dynamically load new values. The modification does not affect the established primary/standby connection and takes effect for subsequent primary/standby replication requests and primary/standby rebuilding requests.
- It supports the standby node rebuild validation under the Quorum and DCF protocols and the primary/standby replication validation under the Quorum protocol. It does not support primary/standby replication validation under the DCF protocol.
- Authentication between the primary and standby database instances is not supported, including primary and standby Dorado instances and DR instances.
- The UUID validation function is used to prevent data crosstalk and pollution caused by incorrect connection between the primary and standby nodes. It is not used for security purposes.
- This parameter cannot be automatically synchronized between the primary and standby nodes.

Value range: enumerated values

- **off**: indicates that UUID validation is disabled.
- **default**: indicates that UUID validation is disabled.
- **uuid**: indicates that UUID validation is enabled.

Default value: **default**

repl_uuid

Parameter description: Specifies the UUID used for primary/standby UUID validation.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

- If UUID validation is enabled on the primary node and a non-null repl_uuid validation code is configured, UUID validation must also be enabled on the standby node and the same repl_uuid validation code must be configured on the standby node. Otherwise, requests for log replication between the primary and standby nodes and standby node rebuilding will be rejected by the primary node.
- The SIGHUP parameter can dynamically load new values. The modification does not affect the established primary/standby connection and takes effect for subsequent primary/standby replication requests and primary/standby rebuilding requests.
- It supports the standby node rebuild validation under the Quorum and DCF protocols and the primary/standby replication validation under the Quorum protocol. It does not support primary/standby replication validation under the DCF protocol.
- Authentication between the primary and standby database instances is not supported, including primary and standby Dorado instances and DR instances.
- The UUID validation function is used to prevent data crosstalk and pollution caused by incorrect connection between the primary and standby nodes. It is not used for security purposes.
- This parameter cannot be automatically synchronized between the primary and standby nodes.

Value range: a string. The value is a string of 0 to 63 case-insensitive letters and digits. It is converted to lowercase letters for storage. An empty string indicates that UUID validation is disabled.

Default value: an empty string

replconninfo1

Parameter description: Specifies the information about the first node to be listened on and authenticated by the current server.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the first node is configured.

Default value: an empty string

replconninfo2

Parameter description: Specifies the information about the second node to be listened on and authenticated by the current server.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the second node is configured.

Default value: an empty string

replconninfo3

Parameter description: Specifies the information about the third node to be listened on and authenticated by the current server.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the third node is configured.

Default value: an empty string

replconninfo4

Parameter description: Specifies the information about the fourth node to be listened on and authenticated by the current server.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the fourth node is configured.

Default value: an empty string

replconninfo5

Parameter description: Specifies the information about the fifth node to be listened on and authenticated by the current server.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the fifth node is configured.

Default value: an empty string

replconninfo6

Parameter description: Specifies the information about the sixth node to be listened on and authenticated by the current server.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the sixth node is configured.

Default value: an empty string

replconninfo7

Parameter description: Specifies the information about the seventh node to be listened on and authenticated by the current server.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the seventh node is configured.

Default value: an empty string

replconninfo8

Parameter description: Specifies the information about the eighth node to be listened on and authenticated by the current server.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the eighth node is configured.

Default value: an empty string

replconninfo9

Parameter description: Specifies the information about the ninth node to be listened on and authenticated by the current server.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the ninth node is configured.

Default value: an empty string

replconninfo10

Parameter description: Specifies the information about the tenth node to be listened on and authenticated by the current server.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the tenth node is configured.

Default value: an empty string

replconninfo11

Parameter description: Specifies the information about the eleventh node to be listened on and authenticated by the current server.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the eleventh node is configured.

Default value: an empty string

replconninfo12

Parameter description: Specifies the information about the twelfth node to be listened on and authenticated by the current server.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the twelfth node is configured.

Default value: an empty string

replconninfo13

Parameter description: Specifies the information about the thirteenth node to be listened on and authenticated by the current server.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the thirteenth node is configured.

Default value: an empty string

replconninfo14

Parameter description: Specifies the information about the fourteenth node to be listened on and authenticated by the current server.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the fourteenth node is configured.

Default value: an empty string

replconninfo15

Parameter description: Specifies the information about the fifteenth node to be listened on and authenticated by the current server.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the fifteenth node is configured.

Default value: an empty string

replconninfo16

Parameter description: Specifies the information about the sixteenth node to be listened on and authenticated by the current server.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the sixteenth node is configured.

Default value: an empty string

replconninfo17

Parameter description: Specifies the information about the seventeenth node to be listened on and authenticated by the current server.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the seventeenth node is configured.

Default value: an empty string

replconninfo18

Parameter description: Specifies the information about the eighteenth node to be listened on and authenticated by the current server.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the eighteenth node is configured.

Default value: an empty string

cross_cluster_replconninfo1

Parameter description: Specifies the information about the local first node to be listened on and authenticated across database instances.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the first node is configured.

Default value: an empty string

cross_cluster_replconninfo2

Parameter description: Specifies the information about the local second node to be listened on and authenticated across database instances.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the second node is configured.

Default value: an empty string

cross_cluster_replconninfo3

Parameter description: Specifies the information about the local third node to be listened on and authenticated across database instances.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the third node is configured.

Default value: an empty string

cross_cluster_replconninfo4

Parameter description: Specifies the information about the local fourth node to be listened on and authenticated across database instances.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the fourth node is configured.

Default value: an empty string

cross_cluster_replconninfo5

Parameter description: Specifies the information about the local fifth node to be listened on and authenticated across database instances.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the fifth node is configured.

Default value: an empty string

cross_cluster_replconninfo6

Parameter description: Specifies the information about the local sixth node to be listened on and authenticated across database instances.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the sixth node is configured.

Default value: an empty string

cross_cluster_replconninfo7

Parameter description: Specifies the information about the local seventh node to be listened on and authenticated across database instances.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the seventh node is configured.

Default value: an empty string

cross_cluster_replconninfo8

Parameter description: Specifies the information about the local eighth node to be listened on and authenticated across database instances.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the eighth node is configured.

Default value: an empty string

available_zone

Parameter description: Specifies the region where the local node is located.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that no information about the node is configured.

Default value: an empty string

enable_availablezone

Parameter description: Specifies whether the local cascaded standby node can connect to standby nodes across AZs.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **true** indicates that the cascaded standby node can only connect to standby nodes in the same AZ.
- **false** indicates that the cascaded standby node can connect to standby nodes across AZs.

Default value: **false**

max_keep_log_seg

Parameter description: Stream control parameter. In logical replication, physical logs are parsed and converted into logical logs locally on the DN. When the number of physical log files that are not parsed is greater than the value of this parameter, stream control is triggered. The value **0** indicates that the stream control function is disabled.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 2147483647.

Default value: **0**

enable_time_report

Parameter description: Specifies whether to record the time consumed by each redo log.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the time when a redo record is generated is recorded.
- **off** indicates that no record is generated.

Default value: **off**

thread_top_level

Parameter description: Increases the priority of the WALWRITERAUXILIARY || WALWRITER || STARTUP ||WALRECEIVER || WAL_NORMAL_SENDER || PGSTAT threads to the highest.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the priority of the preceding threads is increased to the highest.

- **off** indicates that the priority of the preceding threads is not increased.

Default value: **off**

page_work_queue_size

Parameter description: Specifies the length of the blocking queue of each redo worker.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a value ranging from 1 to 100000.

Default value: **4096**

14.3.7.2 Primary Server

synchronous_standby_names

Parameter description: Specifies a comma-separated list of names of potential standby nodes that support synchronous replication.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

- The current synchronous standby node is on the top of the list. If the current synchronous standby node is disconnected, it will be replaced immediately with the next-highest-priority standby node. Name of the next-highest-priority standby node is added to the list.
- The standby node name can be specified by setting the environment variable **PGAPPNAME**.

Value range: a string. If this parameter is set to *****, the name of any standby node that provides synchronous replication is matched. The value can be configured in the following format:

- ANY *num_sync* (*standby_name* [, ...]) [, ANY *num_sync* (*standby_name* [, ...])]
- [FIRST] *num_sync* (*standby_name* [, ...])
- *standby_name* [, ...]

NOTE

- In the preceding command, *num_sync* indicates the number of standby nodes that need to wait for responses from the transaction, *standby_name* indicates the name of the standby node, and FIRST and ANY specify the policies for selecting standby nodes for synchronous replication from the listed servers.
- **ANY N (dn_instanceId1, dn_instanceId2,...)** indicates that any *N* host names in the brackets are selected as the name list of standby nodes for synchronous replication. For example, **ANY 1(dn_instanceId1, dn_instanceId2)** indicates that either of **dn_instanceId1** or **dn_instanceId2** is used as the standby node for synchronous replication.
- **FIRST N (dn_instanceId1, dn_instanceId2, ...)** indicates that the first N primary node names in the brackets are selected as the standby node name list for synchronous replication based on the priority. For example, **FIRST 1 (dn_instanceId1, dn_instanceId2)** indicates that **dn_instanceId1** is selected as the standby node for synchronous replication.
- The meanings of **dn_instanceId1, dn_instanceId2, ...** are the same as those of **FIRST 1 (dn_instanceId1, dn_instanceId2, ...)**.

NOTICE

This parameter is maintained by the CM. If it is manually modified, data may be lost. For details, see "Unified Cluster Management Tool > Features > Automatic Copy Addition and Reduction by Shard > Remarks" in *Tool Reference*.

If you use the gs_guc tool to set this parameter, perform the following operations:

```
gs_guc reload -Z datanode -N @NODE_NAME@ -D @DN_PATH@ -c "synchronous_standby_names='ANY
NODE 1(dn_instanceId1, dn_instanceId2)'";
```

or

```
gs_guc reload -Z datanode -N @NODE_NAME@ -D @DN_PATH@ -c "synchronous_standby_names='ANY
1(AZ1, AZ2)'";
```

Default value: *****

most_available_sync

Parameter description: Specifies whether transactions on the primary node are not blocked due to faults on synchronous standby nodes. For example, if one of the two synchronous standby nodes is faulty and the other is normal, the primary node waits for the normal synchronous standby node instead of being blocked by the faulty synchronous standby node.

For another example, the quorum protocol is executed and one primary node and three synchronous standby nodes are configured by using ANY 2(node1,node2,node3). When node 1 and node 3 are faulty and node 2 is normal, services on the primary node are not blocked.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the primary node is not blocked when all synchronous standby nodes are faulty.

- **off** indicates that the primary node is blocked when all synchronous standby nodes are faulty.

Default value: **off**

keep_sync_window

Parameter description: Specifies the delay for entering the maximum availability mode.

- If **most_available_sync** is set to **on**, when synchronous standby nodes are faulty in primary/standby scenarios and the number of configured synchronous standby nodes is insufficient (for details, see the meaning of **synchonous_standby_name**), setting **keep_sync_window** will retain the maximum protection mode within the time window specified by **keep_sync_window**. That is, committing transactions on the primary node is blocked, delay the primary node to enter the maximum availability mode.
- If synchronous standby nodes recover from faults and the number of synchronous standby nodes meets the configuration requirements, transactions are not blocked.
- You are advised to set **keep_sync_window** to 5s. This prevents the monitoring system from incorrectly reporting network instability.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to *INT_MAX*. The unit is second.

- The value **0** indicates that the **keep_sync_window** is not set, that is, the maximum availability mode is entered directly.
- Other values indicate the size of the timeout window.

Default value: **0**

NOTE

Setting this parameter may affect the RPO. If the primary node is faulty within the configured timeout window, the data generated from the time when the primary node is blocked to the time when the primary node is faulty may be lost.

enable_stream_replication

Parameter description: Specifies whether data and logs are synchronized between the primary and standby nodes.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

- This parameter is used for performance testing in scenarios where data synchronization to standby nodes is enabled and where it is disabled. If this parameter is set to **off**, tests on abnormal scenarios, such as switchover and faults, cannot be performed to prevent inconsistency between the primary and standby nodes.
- This is a restricted parameter, and you are advised not to set it to **off** in normal service scenarios.

Value range: Boolean

- **on** indicates that data and log synchronization between the primary and standby nodes is enabled.
- **off** indicates that data and log synchronization between the primary and standby nodes is disabled.

Default value: **on**

enable_mix_replication

Parameter description: Specifies how WAL files and data are replicated between the primary and standby nodes.

This parameter is an INTERNAL parameter. Its default value is **off** and cannot be modified.

NOTICE

- This parameter cannot be modified in normal service scenarios. That is, mixed replication of the WAL files and data pages is disabled.

Value range: Boolean

- **on** indicates that the WAL file and data page mixed replication mode is enabled.
- **off** indicates that the WAL file and data page mixed replication mode is disabled.

Default value: **off**

vacuum_defer_cleanup_age

Parameter description: Specifies the number of transactions by which **VACUUM** will defer the cleanup of invalid row-store table records. That is, **VACUUM** and **VACUUM FULL** do not clean up deleted tuples immediately.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 1000000. **0** means no delay.

Default value: **0**

data_replicate_buffer_size

Parameter description: Specifies the amount of memory used by queues when the sender sends data pages to the receiver. The value of this parameter affects the buffer size used during the replication from the primary node to the standby node.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 4096 to 1072693248. The unit is KB.

Default value:

128MB (196-core CPU/1536 GB memory, 128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 64-core CPU/512 GB memory, 60-core CPU/480 GB memory, 32-core CPU/256 GB memory, 16-core CPU/128 GB memory, 8-core CPU/64 GB memory, 4-core CPU/32 GB memory); **4MB** (4-core CPU/16 GB memory)

Setting suggestion: Retain the default value. Do not set **data_replicate_buffer_size** to an excessively large or small value. The following condition must be met: **data_replicate_buffer_size** + **segment_buffers** + **shared_buffers** + **wal_buffers** + **temp_buffers** + **maintenance_work_mem** + **work_mem** + **query_mem** + (Standby node) **wal_receiver_buffer_size** < **max_process_memory** < Memory size of the physical machine. If the value of the memory parameter is too large and exceeds the upper limit of the physical memory, the database cannot be started because it cannot allocate sufficient memory.

walsender_max_send_size

Parameter description: Specifies the size of the log or data sending buffer on the primary node.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 8 to *INT_MAX*. The unit is KB.

Default value: **8MB** (that is, 8192 KB)

enable_data_replicate

Parameter description: Specifies how data is synchronized between the primary and standby nodes when the data is imported to a row-store table.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the primary and standby nodes synchronize data using data pages when the data is imported to a row-store table. When **replication_type** is set to **1**, this parameter cannot be set to **on**. If this parameter is set to **on** using the GUC tool, its value will be forcibly changed to **off**.
- **off** indicates that the primary and standby nodes synchronize data using Xlogs when the data is imported to a row-store table.

Default value: **off**

ha_module_debug

Parameter description: Specifies the replication status log of a specific data block during data replication.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the status of each data block is recorded in logs during data replication.
- **off** indicates that the status of each data block is not recorded in logs during data replication.

Default value: **off**

catchup2normal_wait_time

Parameter description: Specifies the maximum duration for the standby node to catch up with the primary node when **most_available_sync** is enabled in primary/standby scenarios. The value of this parameter is an estimate and may be different from the actual value.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from –1 to 10000. The unit is ms.

- The value **–1** indicates that the primary node is blocked until the data catchup on the standby node is complete.
- The value **0** indicates that the primary node is not blocked during the data catchup on the standby node.
- Other values indicate the maximum duration that the primary node is blocked during the data catchup on the standby node. For example, if this parameter is set to **5000**, the primary node is blocked until the data catchup on the standby node is complete in 5s.

Default value: **–1**

check_sync_standby

Parameter description: Specifies whether to enable the standby node check function. After the **synchronous_standby_names** parameter is correctly configured in the primary/standby scenario, if the synchronous standby node is faulty, the write service on the primary node reports a write failure. This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: on or **off**

- **on** indicates that the standby node check is enabled.
- **off** indicates that the standby node check is disabled.

Default value: **off**

NOTE

- This parameter cannot be synchronized in job work and autonomous transactions. Otherwise, the check may not take effect.
- If the standby node check is not configured for a specified user or session and the standby node is faulty when the forcible synchronization commit mode is enabled, the write operation on a table causes the query on the same table by another user or in another session to hang. In this case, you need to recover the standby node or manually terminate the hung client.
- The standby node check function cannot be enabled in scenarios (such as VACUUM ANALYZE) where non-write operations trigger log writing. If the standby node does not meet the requirements for synchronizing configurations to the standby node, services will be hung in this scenario. In this case, you need to manually terminate the services.

sync_config_strategy

Parameter description: Specifies the policy for synchronizing configuration files between the primary node and standby node, and between the standby node and cascaded standby node.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: enumerated values

- **all_node**: If this parameter is set to **all_node** for the primary node, the primary node is allowed to proactively synchronize configuration files to all standby nodes. If this parameter is set to **all_node** for a standby node, the standby node is allowed to send synchronization requests to its primary node, and the standby node is allowed to proactively synchronize configuration files to all cascaded standby nodes. If this parameter is set to **all_node** for a cascaded standby node, the current cascaded standby node is allowed to send synchronization requests to its standby node.
- **only_sync_node**: If this parameter is set to **only_sync_node** for the primary node, the primary node is only allowed to proactively synchronize configuration files to all standby nodes. If this parameter is set to **only_sync_node** for a standby node, the standby node is allowed to send synchronization requests to its primary node, and the standby node is not allowed to proactively synchronize configuration files to all cascaded standby nodes. If this parameter is set to **only_sync_node** for a cascaded standby node, the current cascaded standby node is allowed to send synchronization requests to its standby node.
- **none_node**: If this parameter is set to **none_node** for the primary node, the primary node is not allowed to proactively synchronize configuration files to all standby nodes. If this parameter is set to **none_node** for a standby node, the standby node is not allowed to send synchronization requests to its primary node, and the standby node is allowed to proactively synchronize configuration files to all cascaded standby nodes. If this parameter is set to **none_node** for a cascaded standby node, the current cascaded standby node is not allowed to send synchronization requests to its standby node.

Default value: all_node

hadr_recovery_time_target

Parameter description: Specifies whether the standby database instance completes log writing and replay in streaming DR mode.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 3600. The unit is second.

0 indicates that log flow control is disabled. A value from 1 to 3600 indicates that a standby node can write and replay logs within the period specified by **hadr_recovery_time_target**. This ensures that the logs can be written and replayed within the period specified by **hadr_recovery_time_target** and the standby database instance can be promoted to primary quickly. If this parameter is set to a small value, the performance of the primary node is affected. If it is set to a large value, the log flow is not effectively controlled.

Default value: **0**

hadr_recovery_point_target

Parameter description: Specifies the RPO time allowed for the standby database instance to flush logs to disks in streaming DR mode.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 3600. The unit is second.

0 indicates that log flow control is disabled. A value from 1 to 3600 indicates that the standby node can flush logs to disks within the period specified by **hadr_recovery_point_target**. This ensures that the log difference between the primary and standby database instances is controlled within the period specified by **hadr_recovery_point_target** during the switchover and the standby database instance can be promoted to primary. If this parameter is set to a small value, the performance of the primary node is affected. If it is set to a large value, the log flow is not effectively controlled.

Default value: **0**

hadr_super_user_record_path

Parameter description: Specifies the path for storing encrypted files of the **hadr_disaster** user in the standby database instance in streaming DR mode. This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Modification suggestion: The value is automatically set by the streaming DR password transfer tool and does not need to be manually added.

Value range: a string.

Default value: **NULL**

NOTICE

- In a database instance that contains a primary node, and a standby node, the primary node is a sender relative to the standby node and the standby node is a receiver relative to the primary node.
- The sender actively synchronizes the configuration file to the receiver, and the receiver requests the sender to synchronize the configuration file, which are two independent events, so that the configuration files are synchronized. If you do not want to synchronize configuration files, set this parameter to **none_node** on the receiver. If the sender is a standby node, set this parameter to **none_node** only. If the sender is a primary node, set this parameter to **none_node** when the primary node does not synchronize with any standby node; or set this parameter to **only_sync_node** when the primary node synchronizes with synchronous standby nodes only and does not synchronize with asynchronous standby nodes.
- To be specific, the sender sends a configuration file which directly overwrites the corresponding parameter in the configuration file of the receiver. After the policy for synchronizing configuration files is set, even if you modify configuration parameters of the receiver, the modification does not take effect because the sender immediately overwrites these parameters.
- The following configuration parameters are not synchronized even if the policy for synchronizing configuration files is set: application_name, audit_directory, available_zone, comm_control_port, comm_sctp_port, listen_addresses, log_directory, port, replconninfo1, replconninfo2, replconninfo3, replconninfo4, replconninfo5, replconninfo6, replconninfo7, replconninfo8, replconninfo9, replconninfo10, replconninfo11, replconninfo12, replconninfo13, replconninfo14, replconninfo15, replconninfo16, replconninfo17, replconninfo18, ssl, ssl_ca_file, ssl_cert_file, ssl_ciphers, ssl_crl_file, ssl_key_file, ssl_renegotiation_limit, ssl_cert_notify_time, synchronous_standby_names, local_bind_address, perf_directory, query_log_directory, asp_log_directory, streaming_router_port, enable_upsert_to_merge, recovery_min_apply_delay, and sync_config_strategy.

enable_wal_sender_crc_check

Parameter description: Specifies whether to enable CRC for the WAL sender thread. CRC is performed before the primary node sends logs. If this parameter is enabled, the performance may deteriorate by less than 5%.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the check function is enabled.
- **off** indicates that the check function is disabled.

Default value: **on**

14.3.7.3 Standby Server

hot_standby

Parameter description: Specifies whether to allow connections and queries on a standby node during its recovery.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

- If this parameter is set to **on**, **wal_level** must be set to **hot_standby** or a higher level. Otherwise, the database startup fails.
- In an HA system, **hot_standby** cannot be set to **off**, because this setting can affect other features of the HA system.
- If the **hot_standby** parameter was disabled and the **wal_level** parameter was set to a value smaller than that of **hot_standby**, perform the following operations to ensure that the logs to be replayed on the standby node can be queried on the standby node before enabling the **hot_standby** parameter again:
  1. Change the value of **wal_level** of the primary and standby nodes to the value of **hot_standby** or a higher value, and restart the instances for the change to take effect.
  2. Perform the checkpoint operation on the primary node and query the **pg_stat_get_wal_senders()** function to ensure that the value of **receiver_replay_location** of each standby node is the same as that of **sender_flush_location** of the primary node. Ensure that the value adjustment of **wal_level** is synchronized to the standby nodes and takes effect, and the standby nodes do not need to replay low-level logs.
  3. Set the **hot_standby** parameter of the primary and standby nodes to **on**, and restart the instances for the setting to take effect.
- If the read on standby function is enabled, conflicts between replay and query on the standby node may cause the query to cancel. The error information is as follows:
  - ERROR: canceling statement due to conflict with recovery
  - ERROR: terminating connection due to conflict with recovery
- During the standby node read in serial and parallel replay modes, when the primary node rebuilds indexes online, an error may be reported when the standby node is read. Try again later. The error information is as follows:
  - could not open relation with OID xxx during recovery delete object, please try again later
  - Catalog is missing xxx attribute(s) for relid xxx
  - cache lookup failed for index xxx, refilenode:xxx, name:"xxx"
  - could not find pg_class entry for xxx
- During the standby node read in the case of ultimate RTO, the VM with small memory configuration may report an error indicating insufficient memory. You can enable the GUC parameter **exrto_standby_read_opt** (enabled by default) to effectively reduce the memory and I/O overhead.
- Use sysbench to test the read performance of the standby node in typical scenarios: The primary node executes 100 concurrent update services. The primary and standby nodes both execute 200 concurrent read services at the same time. When the I/O and CPU are not limited, the read performance of the standby node in serial replay mode is not lower than 80% of that of the primary node, and the read performance of the standby node in ultimate RTO deteriorates by no more than 10% compared with that of the standby node in serial replay mode.

Value range: Boolean

- **on** indicates that connections and queries are allowed on the standby node during the recovery.
- **off** indicates that connections and queries are not allowed on the standby node during the recovery.

Default value: **on**

max_standby_archive_delay

Parameter description: Specifies the wait period before queries on a standby node are canceled when the queries conflict with WAL processing and archiving in hot standby mode. In the current version, the setting does not take effect and is controlled by the **max_standby_streaming_delay** parameter.

Parameter type: integer

Unit: millisecond

Value range: 1 to 2147483647

NOTE

–1 indicates that the standby node waits until the conflicting queries are complete.

Default value: **3000ms** (that is, 3s)

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

max_standby_streaming_delay

Parameter description: Specifies the wait period before queries on the standby node are canceled when the queries conflict with WAL data receiving through streaming replication in hot standby mode. If this parameter is set to a large value or the service load is heavy, an error may be reported for waiting for transaction replay and flushing to disks.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

–1 indicates that the standby node waits until the conflicting queries are complete. In the scenario where serial or parallel replay is enabled, if the system detects that the query thread and replayer thread are in the deadlock state, the system still cancels the query to prevent replay blocking.

Value range: an integer ranging from –1 to *INT_MAX*. The unit is ms.

Default value: **3000ms** (that is, 3s)

wal_receiver_status_interval

Parameter description: Specifies the maximum interval for notifying the primary node of the WAL receiver status.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to *INT_MAX*/1000. The unit is s.

Default value: **5s**

NOTICE

If this parameter is set to **0**, the standby node does not send information, such as the log receiving location, to the primary node. As a result, the transaction commit on the primary node may be blocked, and the switchover may fail. In normal service scenarios, you are advised not to set this parameter to **0**.

hot_standby_feedback

Parameter description: Specifies whether the ID of the oldest transaction active on standby nodes will be sent to the primary node, preventing a query conflict.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the ID of the oldest transaction active on standby nodes will be sent to the primary node.
- **off** indicates that the ID of the oldest transaction active on standby nodes will not be sent to the primary node.

Default value: **off**

NOTICE

If this parameter is set to **on**, VACUUM on the primary node will not clean up tuples modified in transactions later than the oldest transaction active on standby nodes.

Therefore, the performance of the primary node will be affected. If replay conflicts with query on the standby node and a query error is reported, you are advised to increase the value of **max_standby_streaming_delay**.

wal_receiver_timeout

Parameter description: Specifies the maximum wait period for a standby node to receive data from the primary node.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to *INT_MAX*. The unit is ms.

Default value: **6s** (that is, 6000 ms)

wal_receiver_connect_timeout

Parameter description: Specifies the timeout interval for a standby node to connect to the primary node.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to *INT_MAX*/1000. The unit is s.

Default value: **2s**

wal_receiver_connect_retries

Parameter description: Specifies the maximum attempts that a standby node connects to the primary node.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to *INT_MAX*

Default value: **1**

wal_receiver_buffer_size

Parameter description: Specifies the memory buffer size for the standby nodes to store the received Xlog files.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 4096 to 1047552. The unit is KB.

Default value: **64MB** (that is, 65536 KB)

primary_slotname

Parameter description: Specifies the slot name of the primary node corresponding to a standby node. This parameter is used for the mechanisms to verify the primary-standby relationship and delete WALs.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string

Default value: an empty string

max_standby_base_page_size

Parameter description: Specifies the maximum storage space of base page files on the standby node after the ultimate RTO function is enabled.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a long integer ranging from 1048576 to 562949953421311. The unit is KB.

Default value: **268435456** (256 GB)

max_standby_lsn_info_size

Parameter description: Specifies the maximum size of LSN info files on the standby node after the ultimate RTO function is enabled.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a long integer ranging from 1048576 to 562949953421311. The unit is KB.

Default value: **268435456** (256 GB)

max_keep_csn_info_size

Parameter description: Specifies the maximum size of CSN info linked lists allowed by each DN on the standby node after the ultimate RTO function is enabled.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a long integer ranging from 16384 to 131072. The unit is KB.

Default value: **16384** (16 MB)

base_page_saved_interval

Parameter description: Specifies the interval for generating base pages on the standby node after the ultimate RTO function is enabled. For the same page, a base page is generated each time the value of this parameter is replayed.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 5 to 2000.

Default value: **400**

standby_force_recycle_ratio

Parameter description: Specifies the percentage of files read by the standby node to trigger forcible recycling after the ultimate RTO function is enabled. When the total size of base page files exceeds the value of **max_standby_base_page_size** x **standby_force_recycle_ratio** or the total size of LSN info files exceeds the value of **max_standby_lsn_info_size** x **standby_force_recycle_ratio**, forcible recycling is triggered and some queries are canceled. When the value of **standby_force_recycle_ratio** is **0**, forcible recycling is not started, and the setting of **max_standby_base_page_size** and **max_standby_lsn_info_size** does not take effect.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a double-precision floating-point number ranging from 0.0 to 1.0

Default value: **0.8**

standby_recycle_interval

Parameter description: Specifies the interval for the standby node to recycle read files after the ultimate RTO function is enabled. The thread for recycling read

resources on the standby node attempts to clear read files on the standby node at the interval specified by this parameter.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 86400. The unit is s.

Default value: **10**

standby_max_query_time

Parameter description: Specifies the maximum query time supported on the standby node after the ultimate RTO function is enabled. If the query time exceeds the value of this parameter, the query will be canceled. Note: The time when the query is canceled is affected by the interval parameter **standby_recycle_interval** of the recycling thread and the time when the snapshot is obtained. Therefore, the actual execution time of the query on the standby node must be greater than the value of this parameter.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 86400. The unit is s.

Default value: **600**

exrto_standby_read_opt

Parameter description: Specifies whether to support read optimization of the standby node with ultimate RTO. This parameter is enabled by default. This parameter is not synchronized between the primary and standby nodes.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean. **on** indicates that the optimization is enabled, and **off** indicates that the optimization is disabled.

Default value: **on**

walrcv_writer_crc_check_level

Parameter description: Specifies whether to enable Xlog verification on the standby node in the primary/standby databases in the streaming DR scenario. By default, the parameter is enabled only in the DR database.

Parameter type: integer

Unit: none

Value range: 0 to 3

- **0**: The verification is disabled.
- **1**: The verification in the DR database takes effect.
- **2**: The verification on the standby node of the primary database takes effect.
- **3**: Standby nodes in the primary database and nodes in the DR database take effect.

Default value: **1**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

14.3.8 Query Planning

This section describes the method configuration, cost constants, planning algorithm, and some configuration parameters for the optimizer.

NOTE

- Two parameters are involved in the optimizer:
  - *INT_MAX* indicates the maximum value of the INT data type. The value is **2147483647**.
  - *DBL_MAX* indicates the maximum value of the FLOAT data type.
- In addition to customer services, global query planning parameters also affect database O&M and monitoring services, such as WDR generation, scale-out, redistribution, and data import and export.

14.3.8.1 Optimizer Method Configuration

These configuration parameters provide a crude method of influencing the query plans chosen by the query optimizer. If the default plan chosen by the optimizer for a particular query is not optimal, a temporary solution is to use one of these configuration parameters to force the optimizer to choose a different plan. Better ways include adjusting the optimizer cost constants, manually running **ANALYZE**, increasing the value of the **default_statistics_target** parameter, and increasing the amount of the statistics collected in specific columns using **ALTER TABLE SET STATISTICS**.

enable_broadcast

Parameter description: Controls the query optimizer's use of broadcast distribution method when it evaluates the cost of stream.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: used.
- **off**: not used.

Default value: **on**

NOTE

This parameter does not take effect in the current version.

enable_bitmapscan

Parameter description: Controls the query optimizer's use of bitmap-scan plan types.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: used.
- **off**: not used.

Default value: **on**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: After this function is disabled, the bitmap scan operator is disabled globally, which may cause performance deterioration in some scenarios. Exercise caution when modifying the parameter to avoid risks caused by misoperations.

force_bitmapand

Parameter description: Specifies whether the optimizer forcibly uses BitmapAnd plan types.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: used.
- **off**: not used.

Default value: **off**

enable_hashagg

Parameter description: Controls the query optimizer's use of Hash aggregation plan types.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: used.
- **off**: not used.

Default value: **on**

enable_hashjoin

Parameter description: Controls the query optimizer's use of Hash-join plan types.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: used.
- **off**: not used.

Default value: **on**

enable_indexscan

Parameter description: Controls the query optimizer's use of index-scan plan types.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: used.
- **off**: not used.

Default value: **on**

enable_indexonlyscan

Parameter description: Controls the query optimizer's use of index-only-scan plan types.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: used.
- **off**: not used.

Default value: **on**

enable_material

Parameter description: Controls the query optimizer's use of materialization. It is impossible to suppress materialization entirely, but setting this variable to **off** prevents the optimizer from inserting materialized nodes.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: used.
- **off**: not used.

Default value: **on**

enable_mergejoin

Parameter description: Controls the query optimizer's use of merge-join plan types.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: used.
- **off**: not used.

Default value: **off**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

enable_nestloop

Parameter description: Controls the query optimizer's use of nested-loop join plan types to fully scan internal tables. It is impossible to suppress nested-loop joins entirely, but setting this variable to **off** encourages the optimizer to choose other methods if available.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: used.
- **off**: not used.

Default value: **off**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

enable_index_nestloop

Parameter description: Controls the query optimizer's use of the index nested-loop join plan types to scan the parameterized indexes of internal tables.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: used.
- **off**: not used.

Default value: **on**

enable_seqscan

Parameter description: Controls the query optimizer's use of sequential scan plan types. It is impossible to suppress sequential scans entirely, but setting this variable to **off** encourages the optimizer to choose other methods if available.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: used.
- **off**: not used.

Default value: **on**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

enable_sort

Parameter description: Controls the query optimizer's use of sort methods. It is impossible to suppress explicit sorts entirely, but setting this variable to **off** encourages the optimizer to choose other methods if available.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: used.
- **off**: not used.

Default value: **on**

enable_tidscan

Parameter description: Controls the query optimizer's use of Tuple ID (TID) scan plan types.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: used.
- **off**: not used.

Default value: **on**

enable_kill_query

Parameter description: In CASCADE mode, when a user is deleted, all the objects belonging to the user are deleted. This parameter specifies whether the queries of the objects belonging to the user can be unlocked when the user is deleted.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the unlocking is allowed.
- **off** indicates that the unlocking is not allowed.

Default value: **off**

enforce_a_behavior

Parameter description: Controls the rule matching modes of regular expressions.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the A matching rule is used.
- **off** indicates that the POSIX matching rule is used.

Default value: **on**

max_recursive_times

Parameter description: Specifies the maximum number of **WITH RECURSIVE** iterations.

Parameter type: integer.

Unit: none

Value range: 0 to *INT_MAX*

Default value: **200**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

enable_change_hjcost

Parameter description: Specifies whether the optimizer excludes inner table running costs when selecting the hash join cost path. If it is set to **on**, tables with a few records and high running costs are more possible to be selected.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: used.
- **off**: not used.

Default value: **off**

enable_absolute_tablespace

Parameter description: Controls whether the tablespace can use an absolute path.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that an absolute path can be used.
- **off** indicates that an absolute path cannot be used.

Default value: **on**

enable_valuepartition_pruning

Parameter description: Specifies whether the DFS partitioned table is dynamically or statically optimized.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the DFS partitioned table is dynamically or statically optimized.
- **off** indicates that the DFS partitioned table is not dynamically or statically optimized.

Default value: **on**

qrw_inlist2join_optmode

Parameter description: Specifies whether to enable inlist-to-join (inlist2join) query rewriting.

Parameter type: string.

Unit: none

Value range:

- **disable** indicates that the inlist2join query rewriting is disabled.
- **cost_base** indicates that the cost-based inlist2join query rewriting is enabled.
- **rule_base** indicates that the rule-based inlist2join query rewriting is forcibly enabled.
- A positive integer (1 to 2147483647) indicates the threshold of inlist2join query rewriting. If the number of elements in the list is greater than the threshold, the rewriting is performed.

Default value: **cost_base**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

skew_option

Parameter description: Specifies whether an optimization policy is used.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string.

- **off** indicates that the policy is disabled.
- **normal** indicates that a radical policy is used. All possible skews are optimized.
- **lazy** indicates that a conservative policy is used. Uncertain skews are ignored.

Default value: **normal**

default_limit_rows

Parameter description: Specifies the default estimated number of limit rows for generating genericplan. If this parameter is set to a non-negative value, the value is used as the estimated number of limit rows. If this parameter is set to a negative value, the value is converted to a percentage and used as default estimated value; for example, **-5** indicates 5%.

Parameter type: floating point

Unit: none

Value range: -100 to *DBL_MAX*

Default value: **–10**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none

check_implicit_conversions

Parameter description: Specifies whether to check candidate index paths generated for index columns that have implicit type conversions in a query.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that a check will be performed for candidate index paths generated for index columns that have implicit type conversion in a query.
- **off** indicates that a check will not be performed.

Default value: **off**

cost_weight_index

Parameter description: Specifies the cost weight of index_scan.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a floating-point number ranging from 1e-10 to 1e+10.

Default value: **1**

enable_default_index_deduplication

Parameter description: Specifies whether to deduplicate and compress tuples with duplicate key values for a B-tree index by default. The deduplication and compression functions do not take effect for primary key indexes and unique indexes. When there are a large number of indexes with duplicate key values, the deduplication and compression function can effectively reduce the space occupied by indexes. In scenarios where non-unique indexes are used and index key values are seldom repeated or unique, the deduplication and compression function slightly deteriorates the index insertion performance. If the WITH (**deduplication** set to **on**/**off**) syntax is used during index creation, the deduplication parameter is preferentially used to determine whether to use deduplication and compression for the index.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **off**: indicates that the deduplication and compression function is disabled for B-tree indexes. This is the default value.

- **on**: indicates that the deduplication and compression function is enabled for B-tree indexes.

Default value: **off**

enable_expr_fusion

Parameter description: Specifies whether to enable the SRF, expression flattening, centralized Seq Scan projection cancellation, transition status of shared aggregate functions, and step number optimization features.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **off**: indicates that this function is disabled. This is the default value.
- **on**: indicates that the SRF, expression flattening, centralized Seq Scan projection cancellation, transition status of shared aggregate functions, and step number optimization features are enabled.

Default value: **off**

NOTE

The SRF supports only the scenario where **query_dop** is set to **1**.

enable_opfusion_reuse

Parameter description: Specifies whether to reuse the opfusion obj memory.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the opfusion obj memory is reused when conditions are met.
- **off** indicates that this function is disabled.

Default value: **on**

enable_iud_fusion

Parameter description: Specifies whether to optimize the IUD statements.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the input is optimized during type conversion.
- **off** indicates that this function is disabled.

Default value: **off**

14.3.8.2 Optimizer Cost Constants

This section describes the optimizer cost constants. The cost variables described here are measured on an arbitrary scale. Only their relative values matter, therefore scaling them all up or down by the same factor will result in no change

in the optimizer's choices. By default, they use the cost of fetching sequential pages as the basic unit. That is, **seq_page_cost** is set to **1.0** and the other cost variables are set with reference to the parameter. However, you can use a different scale, such as actual execution time in milliseconds.

seq_page_cost

Parameter description: Specifies the optimizer's estimated cost of a disk page fetch that is part of a series of sequential fetches.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a floating-point number ranging from 0 to *DBL_MAX*.

Default value: **1**

random_page_cost

Parameter description: Specifies the optimizer's estimated cost of an out-of-sequence disk page fetch.

Parameter type: floating point

Unit: none

Value range: 0 to *DBL_MAX*

Default value: **4**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none

NOTICE

Although the server allows you to set **random_page_cost** to a value less than that of **seq_page_cost**, it is not physically sensitive to do so. However, setting them equal makes sense if the database is entirely cached in RAM, because in that case there is no penalty for fetching pages out of sequence. Also, in a heavily-cached database you should lower both values relative to the CPU parameters, since the cost of fetching a page already in RAM is much smaller than it would normally be.

NOTE

- This value can be overwritten for tables and indexes in a particular tablespace by setting the tablespace parameter of the same name.
- Reducing this value relative to **seq_page_cost** will cause the system to prefer index scans and raising it will make index scans relatively more expensive. You can increase or decrease both values together to change the disk I/O costs relative to CPU costs.

cpu_tuple_cost

Parameter description: Specifies the optimizer's estimated cost of processing each row during a query.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a floating-point number ranging from 0 to *DBL_MAX*.

Default value: **0.01**

cpu_index_tuple_cost

Parameter description: Specifies the optimizer's estimated cost of processing each index entry during an index scan.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a floating-point number ranging from 0 to *DBL_MAX*.

Default value: 0.005

cpu_operator_cost

Parameter description: Specifies the optimizer's estimated cost of processing each operator or function executed during a query.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a floating-point number ranging from 0 to *DBL_MAX*.

Default value: **0.0025**

effective_cache_size

Parameter description: Specifies the optimizer's assumption about the effective size of the disk cache that is available to a single query.

When setting this parameter you should consider both GaussDB's shared buffers and the kernel's disk cache. Also, take into account the expected number of concurrent queries on different tables, since they will have to share the available space.

This parameter has no effect on the size of shared memory allocated by GaussDB. It is used only for estimation purposes and does not reserve kernel disk cache. The value is in the unit of disk page. Usually the size of each page is 8192 bytes.

Parameter type: integer.

Unit: page (8 KB)

Value range:

- Method 1: Set this parameter to an integer without a unit. The integer ranges from 1 to 2147483647. For example, **200** indicates that there are 200 pages, that is, the size of 200 x 8 KB.
- Method 2: Set this parameter to a value with a unit. The value ranges from 1 x 8 KB to 2147483647 x 8 KB. For example, the value **200MB** indicates 200 MB. The unit can only be KB, MB, or GB.

Default value:

280GB (196-core CPU/1536 GB memory); **180GB** (128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory); **135GB**

(96-core CPU/768 GB memory); **100GB** (80-core CPU/640 GB memory); **90GB** (64-core CPU/512 GB memory); **80GB** (60-core CPU/480 GB memory); **40GB** (32-core CPU/256 GB memory); **18GB** (16-core CPU/128 GB memory); **8GB** (8-core CPU/64 GB memory); **4GB** (4-core CPU/32 GB memory); **2GB** (4-core CPU/16 GB memory)

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: A value greater than the default one may enable index scan, and a value less than the default one may enable sequential scan.

allocate_mem_cost

Parameter description: Specifies the query optimizer's estimated cost of creating a hash table for memory space using hash join. This parameter is used for optimization when the hash join estimation is inaccurate.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a floating-point number ranging from 0 to *DBL_MAX*.

Default value: **0**

14.3.8.3 Genetic Query Optimizer

This section describes parameters related to genetic query optimizer. The genetic query optimizer (GEQO) is an algorithm that plans queries by using heuristic searching. This algorithm reduces planning time for complex queries and the costs of producing plans are sometimes inferior to those found by the normal exhaustive-search algorithm.

geqo

Parameter description: Specifies whether to enable the genetic query optimization.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

It is best not to turn it off in execution. **geqo_threshold** provides more subtle control of GEQO.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **on**

geqo_threshold

Parameter description: Specifies the number of **FROM** items. Genetic query optimization is used to plan queries when the number of statements executed is greater than this value.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

- For simpler queries, it is best to use the regular, exhaustive-search planner; but for queries with many tables, it is better to use GEQO to manage the queries.
- A **FULL OUTER JOIN** construct counts as only one **FROM** item.

Value range: an integer ranging from 2 to *INT_MAX*.

Default value: **12**

geqo_effort

Parameter description: Controls the trade-off between planning time and query plan quality in GEQO.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

- **geqo_effort** does not perform operations directly. This parameter is only used to compute the default values for the other variables that influence GEQO behavior. If you prefer, you can manually set the other parameters instead.
- Larger values increase the time spent in query planning, but also increase the probability that an efficient query plan is chosen.

Value range: an integer ranging from 1 to 10.

Default value: **5**

geqo_pool_size

Parameter description: Controls the pool size used by GEQO, that is, the number of individuals in the genetic population.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to *INT_MAX*.

NOTICE

The value of this parameter must be at least **2**, and useful values are typically from **100** to **1000**. If this parameter is set to **0**, GaussDB selects a proper value based on **geqo_effort** and the number of tables.

Default value: **0**

geqo_generations

Parameter description: Specifies the number of iterations of the GEQO.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to *INT_MAX*.

NOTICE

The value of this parameter must be at least **1**, and useful values are typically from **100** to **1000**. If it is set to **0**, a suitable value is chosen based on **geqo_pool_size**.

Default value: **0**

geqo_selection_bias

Parameter description: Specifies the selection bias used by GEQO. The selection bias is the selective pressure within the population.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a floating-point number ranging from 1.5 to 2.0.

Default value: **2**

geqo_seed

Parameter description: Specifies the initial value of the random number generator used by GEQO to select random paths through the join order search space.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a floating-point number ranging from 0.0 to 1.0.

NOTICE

Varying the value changes the set of join paths explored, and may result in a better or worse best path being found.

Default value: **0**

14.3.8.4 Other Optimizer Options

cost_model_version

Parameter description: Specifies the version of the optimizer cost model. It can be regarded as a protection parameter to disable the latest optimizer cost model and keep consistent with the plan of the earlier version. If the value of this parameter is changed, many SQL plans may be changed. Therefore, exercise caution when performing this operation.

Parameter type: integer.

Unit: none

Value range: **0**, **1**, **2**, **3**, or **4**

- **0** indicates that the latest cost estimation model is used. The current version is equivalent to **4**.
- **1** indicates that the original cost estimation model is used.
- **2**: indicates that the enhanced COALESCE expression, hash join cost, and semi/anti join cost are used for estimation on the basis of **1**.
- **3**: indicates that the boundary correction estimator is used to estimate the NDV on the basis of **2**. The hint of indexscan can be applied to indexonlyscan.
- **4**: indicates that partition-level statistics are used for cost estimation on the basis of **3**.

Default value: **0**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: When upgrading the database, you are advised to set this parameter the same as that of the source version. When installing a new environment, you are advised to set this parameter to the default value.

explain_dna_file

Parameter description: Sets **explain_perf_mode** to **run** to export object files in CSV format.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

The value of this parameter must be an absolute path plus a file name with the extension **.csv**.

Value range: a string.

Default value: empty

explain_perf_mode

Parameter description: Specifies the display format of the **explain** command.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: normal, **pretty**, **summary**, and **run**

- **normal** indicates that the default printing format is used.
- **pretty** indicates a new format improved by using GaussDB. The new format contains a plan node ID, directly and effectively analyzing performance.
- **summary** indicates that the analysis result on this information is printed in addition to the printed information specified by **pretty**.
- **run** indicates that the system exports the printed information specified by **summary** as a CSV file for further analysis.

NOTICE

The display sequence may vary greatly according to the display format of **explain**. The examples of the **normal** and **pretty** formats are described as follows:

Example of the **normal** format:

```
                                                  QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=21.23..21.23 rows=1 width=306)
   Sort Key: supplier.s_suppkey
   CTE revenue
     ->  HashAggregate  (cost=12.88..12.88 rows=1 width=76)
           Group By Key: lineitem.l_suppkey
           ->  Partition Iterator  (cost=0.00..12.87 rows=1 width=44)
                 Iterations: 7
                 ->  Partitioned Seq Scan on lineitem  (cost=0.00..12.87 rows=1 width=44)
                       Filter: ((l_shipdate >= '1996-01-01 00:00:00'::timestamp(0) without time zone) AND
(l_shipdate < '1996-04-01 00:00:00'::timestamp without time zone))
                       Selected Partitions:  1..7
   InitPlan 2 (returns $3)
     ->  Aggregate  (cost=0.02..0.03 rows=1 width=64)
           ->  CTE Scan on revenue  (cost=0.00..0.02 rows=1 width=32)
   ->  Nested Loop  (cost=0.00..8.30 rows=1 width=306)
         ->  CTE Scan on revenue  (cost=0.00..0.02 rows=1 width=40)
               Filter: (total_revenue = $3)
         ->  Partition Iterator  (cost=0.00..8.27 rows=1 width=274)
               Iterations: 7
               ->  Partitioned Index Scan using supplier_s_suppkey_idx on supplier  (cost=0.00..8.27 rows=1
width=274)
                     Index Cond: (s_suppkey = revenue.supplier_no)
                     Selected Partitions:  1..7
(21 rows)
```

Example of the **pretty** format:

```
 id |                      operation                                    | E-rows | E-width |    E-costs
----+-------------------------------------------------------------------+--------+---------+----------------
  1 | ->  Sort                                                          |      1 |     306 | 21.230..21.235
  2 |    ->  Nested Loop (3,9)                                          |      1 |     306 | 0.000..8.303
  3 |       ->  CTE Scan on revenue                                     |      1 |      40 | 0.000..0.022
  4 |    ->  HashAggregate  [3, CTE revenue]                            |      1 |      76 | 12.875..12.885
  5 |       ->  Partition Iterator                                      |      1 |      44 | 0.000..12.865
  6 |          ->  Partitioned Seq Scan on lineitem                     |      1 |      44 | 0.000..12.865
  7 |    ->  Aggregate  [4, InitPlan 2 (returns $3)]                    |      1 |      64 | 0.022..0.033
  8 |       ->  CTE Scan on revenue                                     |      1 |      32 | 0.000..0.020
  9 |       ->  Partition Iterator                                      |      1 |     274 | 0.000..8.270
 10 |          ->  Partitioned Index Scan using supplier_s_suppkey_idx on supplier |      1 |     274 | 0.000..8.270
(10 rows)

                                      Predicate Information (identified by plan id)
------------------------------------------------------------------------------------------------------------------------------------------------------------
   5 --Partition Iterator
         Iterations: 7
   6 --Partitioned Seq Scan on lineitem
         Filter: ((l_shipdate >= '1996-01-01 00:00:00'::timestamp(0) without time zone) AND (l_shipdate <
'1996-04-01 00:00:00'::timestamp without time zone))
         Selected Partitions:  1..7
   3 --CTE Scan on revenue
         Filter: (total_revenue = $3)
   9 --Partition Iterator
         Iterations: 7
  10 --Partitioned Index Scan using supplier_s_suppkey_idx on supplier
         Index Cond: (s_suppkey = revenue.supplier_no)
         Selected Partitions:  1..7
(12 rows)
```

Note: The plan blocks in the preceding two formats are different display formats of the same plan. In the **pretty** format, the parts in bold are the CET and InitPlan plan blocks, which may be inserted in the middle of the join block. When the join block is being read, skip the CTE and InitPlan blocks to find the inner table of the corresponding join block.

Default value: pretty

analysis_options

Parameter description: Specifies whether to enable function options in the corresponding options to use the corresponding location functions, including data verification and performance statistics. For details, see the options in the value range.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string.

- **LLVM_COMPILE** indicates that the codegen compilation time of each thread is displayed on the explain performance page.
- **HASH_CONFLICT** indicates that the log in the **gs_log** directory of the database node process displays the statistics of the hash table, including the hash table size, hash link length, and hash conflict.
- **STREAM_DATA_CHECK** indicates that a CRC check is performed on data before and after network data transmission.

Default value:
ALL,on(),off(LLVM_COMPILE,HASH_CONFLICT,STREAM_DATA_CHECK), which indicates that no location function is enabled.

cost_param

Parameter description: Specifies use of different estimation methods in specific customer scenarios, allowing estimated values approximating to onsite values. This parameter can control various methods simultaneously by performing AND (&) on the bit of each method. A method is selected if the result value is not **0**.

When **cost_param & 1** is set to a value other than **0**, an improved mechanism is used for connecting the selectivity of non-equi-joins. This method is more accurate for estimating the selectivity of joins between two identical tables. At present, **cost_param & 1=0** is not used. That is, a better formula is selected for calculation.

When **cost_param & 2** is set to a value other than **0**, the selectivity is estimated based on multiple filter criteria. The lowest selectivity among all filter criteria, but not the product of the selectivities for two tables under a specific filter criterion, is used as the total selectivity. This method is more accurate when a close correlation exists between the columns to be filtered.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to *INT_MAX*.

Default value: **0**

var_eq_const_selectivity

Parameter description: Determines whether to use the new selectivity model to estimate the integer const selectivity.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the new selectivity model is used to calculate the selectivity of the integer const.
  - If an integer does not fall into the MCV, is not NULL, but falls into the histogram, the left and right boundaries of the histogram are used for estimation. If the integer does not fall into the histogram, the number of rows in the table is used for estimation.
  - If the integer is NULL or falls into the MCV, the original logic is used to calculate the selectivity.
- **off** indicates that the original selectivity calculation model is used.

Default value: **off**

enable_partitionwise

Parameter description: Specifies whether to select an intelligent algorithm for joining partitioned tables.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that an intelligent algorithm is selected.
- **off** indicates that an intelligent algorithm is not selected.

Default value: **off**

partition_page_estimation

Parameter description: Determines whether to optimize the estimation of partitioned table pages based on the pruning result.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the pruning result is used to optimize the page estimation.
- **off** indicates that the pruning result is not used to optimize the page estimation.

Default value: **off**

partition_iterator_elimination

Parameter description: Determines whether to eliminate the partition iteration operator to improve execution efficiency when the partition pruning result of a partitioned table is a partition.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the partition iteration operator is eliminated.
- **off** indicates that the partition iteration operator is not eliminated.

Default value: **off**

enable_partition_pseudo_predicate

Parameter description: Specifies whether to rewrite pseudo-predicates to calculate the selectivity of query in a specified partition.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that pseudo-predicate rewriting is used.
- **off** indicates that pseudo-predicate rewriting is not used.

Default value: **off**

enable_functional_dependency

Parameter description: Determines whether the statistics about multiple columns generated by ANALYZE contain function dependency statistics and whether the function dependency statistics are used to calculate the selectivity.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates the following functions: 1. The statistics about multiple columns generated by ANALYZE contain function dependency statistics. 2. Function dependency statistics are used to calculate the selectivity.
- **off** indicates the following functions: 1. The statistics about multiple columns generated by ANALYZE do not contain function dependency statistics. 2. Function dependency statistics are not used to calculate the selectivity.

Default value: **off**

rewrite_rule

Parameter description: Specifies the optional query rewriting rules that are enabled. Some query rewriting rules are optional. Enabling them cannot always improve the query efficiency. In a specific customer scenario, you can set the query rewriting rules through this GUC parameter to achieve optimal query efficiency.

This parameter can control the combination of query rewriting rules, for example, there are multiple rewriting rules: **rule1**, **rule2**, **rule3**, and **rule4**. You can perform the following settings:

```
set rewrite_rule=rule1;          -- Enable query rewriting rule rule1
set rewrite_rule=rule2, rule3;     -- Enable query rewriting rules rule2 and rule3
set rewrite_rule=none;         -- Disable all optional query rewriting rules
```

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string

- **none**: No optional query rewriting rules are used.
- **lazyagg**: The Lazy Agg query rewriting rules are used to eliminate aggregation operations in subqueries.
- **magicset**: The Magic Set query rewriting rules are used to associate subqueries which have aggregation operators with the main query in advance to reduce repeated scanning of sublinks.
- **uniquecheck**: The Unique Check query rewriting rules are used to optimize the subquery statements in target columns without agg and check whether the number of returned rows is 1.
- **intargetlist**: The In Target List query rewriting rules are used to improve subqueries in the target column.
- **predpushnormal**: The Predicate Push query rewriting rules are used to push the predicate condition to the subquery.
- **predpushforce**: The Predicate Push query rewriting rules are used to push down predicate conditions to subqueries and use indexes as much as possible for acceleration.
- **predpush**: The optimal plan is selected based on the cost in **predpushnormal** and **predpushforce**.
- **disable_pullup_expr_sublink**: The optimizer is not allowed to pull up sublinks of the expr_sublink type. For details about sublink classification and pull-up principles, see "SQL Tuning Guide > Typical SQL Tuning Methods > Subquery Tuning" in *Developer Guide*.
- **enable_sublink_pullup_enhanced**: Enhanced sublink query rewriting rules are used, including unrelated sublink pull-up of the WHERE and HAVING clauses and WinMagic rewriting optimization.
- **disable_pullup_not_in_sublink**: The optimizer is not allowed to pull up sublinks related to NOT IN. For details about sublink classification and pull-up principles, see "SQL Tuning Guide > Typical SQL Tuning Methods > Subquery Tuning" in *Developer Guide*.
- **disable_rownum_pushdown**: The filter criterion ROWNUM in the parent query cannot be pushed down to the subquery.
- **disable_windowagg_pushdown**: The filter criterion of the window function in the parent query cannot be pushed down to the subquery.

Default value: **magicset**

NOTE

The **partialpush** and **disablerep** parameters can be set but do not take effect.

enable_pbe_optimization

Parameter description: Specifies whether the optimizer optimizes the query plan for statements executed in Parse Bind Execute (PBE) mode.

Parameter type: Boolean.

Unit: none

Value range:

- **on** indicates that the optimizer optimizes the query plan for statements executed in PBE mode.
- **off** indicates that the optimization is not performed.

Default value: **off**

Setting method: This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

enable_global_plancache

Parameter description: Specifies whether to share the cache for the execution plans of statements in PBE queries and stored procedures. Enabling this function can reduce the memory usage of database nodes in high concurrency scenarios.

When **enable_global_plancache** is enabled, the value of **local_syscache_threshold** cannot be less than 16 MB to ensure that GPC takes effect. If the value of **local_syscache_threshold** is less than 16 MB, set it to 16 MB. If the value is greater than 16 MB, the actual value is used.

Parameter type: Boolean.

Unit: none

Value range:

- **on** indicates that cache sharing is enabled for the execution plans of statements in PBE queries and stored procedures.
- **off** indicates no sharing.

Default value: **off**

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

gpc_clean_timeout

Parameter description: When **enable_global_plancache** is set to **on**, if a plan in the shared plan list is not used within the period specified by **gpc_clean_timeout**, the plan will be deleted. This parameter is used to control the retention period of a shared plan that is not used.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 300 to 86400

- The unit is second.

Default value: **1800**, that is, 30 minutes

enable_global_stats

This parameter has been discarded in the current version. Do not set it.

enable_opfusion

Parameter description: Specifies whether to optimize simple addition, deletion, modification, and query operations.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

The restrictions on simple queries are as follows:

- Only index scan and index-only scan are supported, and the filter criteria of all WHERE statements are on indexes.
- Only single tables can be added, deleted, modified, and queried. JOIN and USING operations are not supported.
- Only row-store tables are supported. Partitioned tables and tables with triggers are not supported.
- Information statistics features of active SQL statements and queries per second (QPS) are not supported.
- Tables that are being scaled out or in are not supported.
- System columns cannot be queried or modified.
- Only simple SELECT statements are supported. For example:
  ```
  SELECT c3 FROM t1 WHERE c1 = ? and c2 =10;
  ```
  Only columns in the target table can be queried. Columns **c1** and **c2** are index columns, which can be followed by constants or parameters. You can use **for update**.
- Only simple INSERT statements are supported. For example:
  ```
  INSERT INTO t1 VALUES (?,10,?);
  ```
  Only one VALUES is supported. The type in VALUES can be a constant or a parameter. RETURNING is not supported.
- Only simple DELETE statements are supported. For example:
  ```
  DELETE FROM t1 WHERE c1 = ? and c2 = 10;
  ```
  Columns **c1** and **c2** are index columns, which can be followed by constants or parameters.
- Only simple UPDATE statements are supported. For example:
  ```
  UPDATE t1 SET c3 = c3+? WHERE c1 = ? and c2 = 10;
  ```
  The values modified in column **c3** can be constants, parameters, or a simple expression. Columns **c1** and **c2** are index columns, which can be followed by constants or parameters.

Value range: Boolean

- **on**: used.
- **off**: not used.

Default value: **on**

enable_plsql_opfusion

Parameter description: Optimizes simple ADD, DELETE, MODIFY, and QUERY statements in stored procedures to improve SQL execution performance.

For details about restrictions on simple ADD, DELETE, MODIFY, and QUERY statements, see **enable_opfusion**.

> NOTE
>
> This parameter takes effect only when **enable_opfusion** is enabled.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: used.
- **off**: not used.

Default value: **on**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

sql_beta_feature

Parameter description: Specifies the SQL engine's optional beta features to be enabled, including optimization of row count estimation and query equivalence estimation.

These optional features provide optimization for specific scenarios, but performance deterioration may occur in some scenarios for which testing is not performed. In a specific customer scenario, you can set the query rewriting rules through this GUC parameter to achieve optimal query efficiency.

This parameter determines the combination of the SQL engine's beta features, for example, **feature1**, **feature2**, **feature3**, and **feature4**. You can perform the following settings:

```
-- Enable beta feature feature1 of the SQL engine.
set sql_beta_feature=feature1;
-- Enable beta features feature2 and feature3 of the SQL engine.
set sql_beta_feature=feature2,feature3;
-- Disable all optional beta features of the SQL engine.
set sql_beta_feature=none;
```

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string

- **none**: uses none of the beta optimizer features.
- **sel_semi_poisson**: uses Poisson distribution to calibrate the equivalent semi-join and anti-join selectivity.
- **sel_expr_instr**: uses the matching row count estimation method to provide more accurate estimation for **instr(col, 'const') > 0, = 0, = 1**.
- **param_path_gen**: generates more possible parameterized paths.
- **rand_cost_opt**: optimizes the random read cost of tables that have a small amount of data.
- **param_path_opt**: uses the bloating ratio of the table to optimize the analysis information of indexes.
- **page_est_opt**: optimizes the **relpages** estimation for the analysis information of table indexes.

- **no_unique_index_first**: disables optimization of the primary key index scan path first.
- **join_sel_with_cast_func**: supports type conversion functions when the number of join rows is estimated.
- **canonical_pathkey**: The regular path key is generated in advance. (**pathkey**: a set of ordered key values of data.)

  WARNING

  After this parameter is enabled, the semantics of the output data of statements such as ORDER BY may be different from that of the standard ones in the case of outer join. Contact Huawei engineers to determine whether to enable this parameter.

- **index_cost_with_leaf_pages_only**: specifies whether index leaf nodes are included when the index cost is estimated.
- **a_style_coerce**: enables the Decode type conversion rule to be compatible with O. For details, see the part related to case processing in ORA compatibility mode in "SQL Reference > Type Conversion > UNION, CASE, and Related Constructs" in *Developer Guide*.
- **predpush_same_level**: enables the **predpush** hint to control parameterized paths at the same layer.
- **enable_plsql_smp**: enables parallel execution of queries in stored procedures. Currently, only one query can be executed in parallel at a time, and no parallel execution plan is generated for autonomous transactions and queries in exceptions.
- **disable_bitmap_cost_with_lossy_pages**: disables the computation of the cost of lossy pages in the bitmap path cost.
- **enable_upsert_execute_gplan**: allows execution through **gplan** in the PBE scenario, if the UPDATE clause in the ON DUPLICATE KEY UPDATE statement contains parameters.
- **disable_merge_append_partition**: disables the generation of the Merge Append path for partitioned tables.
- **disable_text_expr_flatten**: disables the function of automatically inlining expressions during comparison between text and numeric types (numeric, bigint).

Default value:
"sel_semi_poisson,sel_expr_instr,rand_cost_opt,param_path_opt,page_est_opt"

default_statistics_target

Parameter description: Specifies the default statistics target for table columns without a column-specific target set by running **ALTER TABLE SET STATISTICS**. The number of rows sampled during statistics collection is affected.

If this parameter is set to a positive number, the number of rows sampled in the statistics histogram is **default_statistics_target** x 300. If the parameter is set to a negative number, it indicates the percentage of statistics collected. The negative number corresponds to a percentage, for example, **–5** means 5%. That is, the

number of sampled rows is the total number of rows multiplied by 5%. This parameter affects only the target number of sampled rows in the statistics. The actual number of sampled rows is also affected by the memory parameter **maintenance_work_mem**.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from –100 to 10000.

NOTICE

- A larger positive number than the default value increases the time required to do **ANALYZE**, but might improve the quality of the optimizer's estimates.
- Changing settings of this parameter may result in performance deterioration. If query performance deteriorates, you can:
  1. Restore to the default statistics.
  2. Use hints to force the optimizer to use the optimal query plan. For details, see "SQL Tuning Guide > Tuning Using Plan Hints" in *Developer Guide*.

Default value: **100**

auto_statistic_ext_columns

Parameter description: Collects statistics about multiple columns based on the first *K* columns of the composite index in the data table. This GUC parameter indicates *K*. For example, if a composite index is (a,b,c,d,e) and the GUC parameter is set to **3**, statistics about multiple columns are generated on columns (a,b) and (a,b,c). Multi-column statistics can make the optimizer estimate the cardinality more accurate when querying with combined conditions.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

- The system catalog does not take effect.
- The statistics take effect only when the types of all columns support the comparison functions '=' and '<'.
- System pseudocolumns in indexes, such as **tableoid** and **ctid**, are not collected.
- By default, distinct values, MCVs without NULL, and MCVs with NULL are collected. If the AI-based cardinality estimation parameter **enable_ai_stats** is enabled, MCVs are not collected. Instead, models for AI-based cardinality estimation are collected.
- If the index for creating multi-column statistics is deleted and no other index contains the multi-column combination, the multi-column statistics will be deleted in the next ANALYZE operation.
- If the value of this parameter decreases, the new index generates multi-column statistics based on the value of this parameter. The generated multi-column statistics that exceed the value of this parameter will not be deleted.
- If you want to disable the multi-column statistics on a specific combination only, you can retain the value of this parameter and run the DDL command **ALTER TABLE** *tablename* **disable statistics** *((column list))* to disable the statistics on multiple columns in a specific combination.

Value range: an integer ranging from 1 to 4. The value **1** indicates that statistics about multiple columns are not automatically collected.

Default value: **1**

constraint_exclusion

Parameter description: Specifies the query optimizer's use of table constraints to optimize queries.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: enumerated values

- **on** indicates that constraints for all tables are examined.
- **off** indicates that constraints for any table are not examined.
- **partition** indicates that only constraints for inheritance child tables and **UNION ALL** subqueries are examined.

NOTICE

When **constraint_exclusion** is set to **on**, the optimizer compares query conditions with the table's **CHECK** constraints, and omits scanning tables for which the conditions contradict the constraints.

Default value: **partition**

NOTE

Currently, **constraint_exclusion** is enabled by default only for cases that are often used to implement table partitioning. Turning this feature on for all tables imposes extra planning on simple queries, and provides no benefit for simple queries. If you have no partitioned tables, set it to **off**.

cursor_tuple_fraction

Parameter description: Specifies the optimizer's estimated fraction of a cursor's rows that are retrieved.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a floating-point number ranging from 0.0 to 1.0.

NOTICE

Smaller values of this setting bias the optimizer towards using **fast start** plans for cursors, which will retrieve the first few rows quickly while perhaps taking a long time to fetch all rows. Larger values put more emphasis on the total estimated time. At the maximum setting of **1.0**, cursors are planned exactly like regular queries, considering only the total estimated time and how soon the first rows might be delivered.

Default value: **0.1**

from_collapse_limit

Parameter description: Specifies whether the optimizer merges sub-queries into upper queries based on the resulting FROM list. The optimizer merges sub-queries into upper queries if the resulting FROM list would have no more than this many items.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to *INT_MAX*.

NOTICE

Smaller values reduce planning time but may lead to inferior execution plans.

Default value: **8**

join_collapse_limit

Parameter description: Specifies whether the optimizer rewrites JOIN constructs (except FULL JOINS) into lists of FROM items based on the number of the items in the result list.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to *INT_MAX*.

NOTICE

- Setting this parameter to **1** prevents join reordering. As a result, the join order specified in the query will be the actual order in which the relations are joined. The query optimizer does not always choose the optimal join order. Therefore, advanced users can temporarily set this variable to **1**, and then specify the join order they desire explicitly.
- Smaller values reduce planning time but lead to inferior execution plans.

Default value: **8**

plan_mode_seed

Parameter description: This is a debugging parameter. Currently, it supports only **OPTIMIZE_PLAN** and **RANDOM_PLAN**. The value **0** (for **OPTIMIZE_PLAN**) indicates the optimized plan using the dynamic planning algorithm. Other values are for **RANDOM_PLAN**, which indicates that the plan is randomly generated. **–1** indicates that users do not specify the value of the seed identifier. In this case, the optimizer generates a random integer from 1 to 2147483647 and a random execution plan based on the generated integer. A GUC parameter value from 1 to 2147483647 is regarded as the seed identifier, based on which the optimizer generates a random execution plan.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from –1 to 2147483647.

Default value: **0**

NOTICE

- If this parameter is set to a random execution plan, the optimizer generates a random execution plan that may not be the optimal one. Therefore, to guarantee the query performance, the default value **0** is recommended during upgrade, scale-out, scale-in, and O&M.
- If this parameter is not set to **0**, the specified plan hint will not be used.

hashagg_table_size

Parameter description: Specifies the hash table size during the execution of the HASH JOIN operation.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to *INT_MAX*/2.

Default value: **0**

enable_codegen

Parameter description: Specifies whether code optimization can be enabled. Currently, the code optimization uses the LLVM optimization.

Parameter type: Boolean.

Unit: none

Value range:

- **on** indicates that code optimization is enabled.
- **off** indicates that code optimization is disabled.

Default value: **on**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

codegen_compile_thread_num

Parameter description: Specifies the number of Codegen compilation threads.

Parameter type: integer.

Unit: none

Value range: 1 to 8

Default value: **1**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. If the number of threads is too large, the system performance may deteriorate. However, when there are a large number of concurrent services, you can increase the number of threads to improve the throughput performance.

llvm_max_memory

Parameter description: Specifies the maximum memory occupied by IRs (including cached and in-use IRs) generated during Codegen compilation. The memory used by Codegen is not applied for by preoccupation. It is a part of **max_dynamic_memory** and is restricted by the **llvm_max_memory** parameter.

Parameter type: integer.

Unit: KB

Value range: 0 to 2147483647. If the value exceeds the specified value, the original recursive execution logic, instead of the Codegen execution logic, is used. When the upper limit is reached and a downgrade is triggered, decreasing the value of **llvm_max_memory** cannot immediately release the memory occupied by extra IRs. The memory occupied by IRs is released after the corresponding SQL statements are executed.

Default value: **131072kB** (128 MB)

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

- If the parameter is set to an excessively small value, the system does not use the Codegen execution logic, affecting the use of functions.
- If the parameter is set to an excessively small value, LLVM compilation may occupy too many resources of other threads. As a result, the overall system performance deteriorates.

enable_codegen_print

Parameter description: Specifies whether the LLVM IR function can be printed in logs.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the IR function can be printed in logs.
- **off** indicates that the IR function cannot be printed in logs.

Default value: **off**

codegen_cost_threshold

Parameter description: The LLVM compilation takes some time to generate executable machine code. Therefore, LLVM compilation is beneficial only when the actual execution cost is more than the sum of the code required for generating machine code and the optimized execution cost. Parameter **codegen_cost_threshold** specifies a threshold. If the estimated execution cost exceeds the threshold, LLVM optimization is performed. codegen uses **plan_rows** of the execution operator as the cost to compare with the value of **codegen_cost_threshold**. You can run the **explain** command to view the value of **plan_rows**.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 2147483647.

Default value: **100000**

enable_bloom_filter

Parameter description: Specifies whether the BloomFilter optimization can be used. This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the BloomFilter optimization can be used.
- **off** indicates that the BloomFilter optimization cannot be used.

Default value: **on**

enable_extrapolation_stats

Parameter description: Specifies whether the extrapolation logic is used for data of DATE type based on historical statistics. The logic can increase the accuracy of estimation for tables whose statistics are not collected in time, but will possibly

provide an overlarge estimation due to incorrect extrapolation. Enable the logic only in scenarios where the data of DATE type is periodically inserted. This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the extrapolation logic is used.
- **off** indicates that the extrapolation logic is not used.

Default value: **off**

autoanalyze

Parameter description: Specifies whether to automatically collect statistics on tables that have no statistics when a plan is generated. **autoanalyze** cannot be used for foreign or temporary tables. To collect statistics, manually perform the ANALYZE operation. If an exception occurs in the database during the execution of autoanalyze on a table, after the database is recovered, the system may still prompt you to collect the statistics of the table when you run the statement again. In this case, manually perform the ANALYZE operation on the table to synchronize statistics.

Parameter type: Boolean.

Unit: none

Value range:

- **on** indicates that the table statistics are automatically collected.
- **off** indicates that the table statistics are not automatically collected.

Default value: **off**

Setting method: This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

NOTE

This parameter does not take effect in centralized mode.

enable_analyze_check

Parameter description: Specifies whether it is allowed to check whether statistics were collected about tables whose **reltuples** and **relpages** are displayed as **0** in **pg_class** during plan generation.

Parameter type: Boolean.

Unit: none

Value range:

- **on** indicates that the tables will be checked.
- **off** indicates that the tables will not be checked.

Default value: **off**

Setting method: This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

enable_sonic_hashagg

Parameter description: Specifies whether to use the hash aggregation operator designed for column-oriented hash tables when certain constraints are met.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the hash aggregation operator designed for column-oriented hash tables is used when certain constraints are met.
- **off** indicates that the hash aggregation operator designed for column-oriented hash tables is not used.

NOTE

- If **enable_sonic_hashagg** is enabled and the hash aggregation operator designed based on the column-oriented hash table is used when the query meets the constraint condition, the memory usage of the hash aggregation operator can be reduced. However, in scenarios where **enable_codegen** is enabled and the performance is significantly improved, the performance of the operator may deteriorate.
- If **enable_sonic_hashagg** is enabled and the hash aggregation operator designed based on the column-oriented hash table is used when the query meets the constraint condition, the operator is displayed as Sonic Hash Aggregation in the execution plan and execution information of Explain Analyze/Performance; when the query does not meet the constraint condition, the operator is displayed as Hash Aggregation. For details, see "SQL Tuning Guide > Introduction to the SQL Execution Plan > Description" in *Developer Guide*.

Default value: **on**

enable_sonic_hashjoin

Parameter description: Specifies whether to use the hash join operator designed for column-oriented hash tables when certain constraints are met.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the hash join operator designed for column-oriented hash tables is used when certain constraints are met.
- **off** indicates that the hash join operator designed for column-oriented hash tables is not used.

NOTE

- Currently, the parameter can be used only for Inner Join.
- If **enable_sonic_hashjoin** is enabled, the memory usage of query using the Hash Inner operator can be reduced. However, in scenarios where the code generation technology can significantly improve performance, the performance of the operator may deteriorate.
- If **enable_sonic_hashjoin** is enabled and the hash join operator designed based on the column-oriented hash table is used when the query meets the constraint condition, the operator is displayed as Sonic Hash Join in the execution plan and execution information of Explain Analyze/Performance; when the query does not meet the constraint condition, the operator is displayed as Hash Join. For details, see "SQL Tuning Guide > Introduction to the SQL Execution Plan > Description" in *Developer Guide*.

Default value: **on**

enable_sonic_optspill

Parameter description: Specifies whether to optimize the number of files to be written to disks for the hash join operator designed for column-oriented hash tables. If this parameter is enabled, the number of files written to disks does not increase significantly when the hash join operator writes a large number of files to disks.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the number of files to be written to disks for the hash join operator designed for column-oriented hash tables is optimized.
- **off** indicates that the number of files to be written to disks for the hash join operator designed for column-oriented hash tables is not optimized.

Default value: **on**

plan_cache_mode

Parameter description: Specifies the policy for generating an execution plan in the prepared statement.

Parameter type: enumerated type

Unit: none

Value range:

- **auto** indicates that the **custom plan** or **generic plan** is selected by default.
- **force_generic_plan** indicates that the generic plan is forcibly used (soft parsing). The generic plan is a plan generated after you run a prepared statement. The plan policy binds parameters to the plan when you run the **EXECUTE** statement to execute the plan. The advantage of this scheme is that repeated optimizer overheads can be avoided in each execution. The disadvantage is that the plan may not be optimal when data skew occurs for the bound parameters and may result in poor plan execution performance. The bound parameters bind the types of parameters transferred for the first time. If the type of a parameter transferred into the same placeholder is different from the previous time, an error is reported.

- **force_custom_plan** indicates that the custom plan is forcibly used (hard parsing). The custom plan is a plan generated after you run a prepared statement where parameters in the EXECUTE statement are embedded. The custom plan generates a plan based on specific parameters in the EXECUTE statement. This scheme generates a preferred plan based on specific parameters each time and has good execution performance. The disadvantage is that the plan needs to be regenerated before each execution, resulting in a large amount of repeated optimizer overhead.

NOTE

This parameter is valid only for prepared statements. It is used when the parameterized field in a prepared statement has severe data skew.

Default value: **auto**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Set this parameter based on the actual service scenario.

enable_hypo_index

Parameter description: Determines whether the optimizer creates virtual indexes when executing the **EXPLAIN** command.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: A virtual index is created when the **EXPLAIN** command is executed.
- **off**: No virtual index is created when the **EXPLAIN** command is executed.

Default value: **off**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

enable_auto_explain

Parameter description: Specifies whether to enable the function of automatically printing execution plans. This parameter is used to locate slow stored procedures or slow queries. It is valid for the currently connected database primary node and directly connected standby node.

Parameter type: Boolean.

Unit: none

Value range:

- **true**: enabled.
- **false**: disabled.

Default value: **false**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. If you want to view the execution plan, enable this parameter. However, this causes the current system performance to deteriorate.

auto_explain_level

Parameter description: Specifies the log level for automatically printing execution plans.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: enumerated type. The value can be **LOG** or **NOTICE**.

- **LOG**: Execution plans are printed as logs.
- **NOTICE**: Execution plans are printed as notices.

Default value: **LOG**

auto_explain_log_min_duration

Parameter description: Specifies the minimum duration of execution plans that are automatically printed. Only execution plans whose duration is greater than the value of **auto_explain_log_min_duration** will be printed. For example, if this parameter is set to **0**, all executed plans are printed. If this parameter is set to **3000**, all executed plans are printed if the execution of a statement takes more than 3000 ms.

Parameter type: integer.

Unit: millisecond

Value range: 0 to 2147483647

Default value: **0**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

query_dop

Parameter description: Specifies the user-defined degree of parallelism (DOP). If the SMP function is enabled, the system uses the specified degree of parallelism.

Parameter type: integer.

Unit: none

Value range: 1 to 64. **1** indicates that parallel query is disabled.

Default value: **1**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

NOTE

After enabling concurrent queries, ensure you have sufficient CPU, memory, and network to achieve the optimal performance.

enable_startwith_debug

Parameter description: Specifies whether to enable the **start with** or **connect by** parameter for debugging. If this parameter is enabled, information about all tail columns related to the **start with** feature is displayed.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean. The value **true** indicates that the function is enabled, and the value **false** indicates that the function is disabled.

Default value: **false**

enable_inner_unique_opt

Parameter description: Specifies that Inner Unique is optimized for nested loop join, hash join, and sort merge join. That is, the number of matching times is reduced when the attribute corresponding to the inner table in the join condition meets the uniqueness constraint.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: used.
- **off**: not used.

Default value: **on**

enable_indexscan_optimization

Parameter description: Specifies whether to optimize B-tree index scan (IndexScan and IndexOnlyScan) in the Astore engine.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: used.
- **off**: not used.

Default value: **on**

immediate_analyze_threshold

Parameter description: Specifies the threshold for automatically analyzing inserted data. When the amount of data inserted at a time reaches the original data amount multiplied by the value of **immediate_analyze_threshold** and the total number of rows exceeds 100, ANALYZE is automatically triggered.

Parameter type: integer.

Unit: none

Value range: 0 to 1000. If this parameter is set to **0**, this function is disabled.

Default value: **0**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Set this parameter to a small value for tables whose data changes rapidly and whose statistics need to be updated continuously. Set this parameter to a large value for tables whose statistics fluctuate greatly only after a certain amount of data is reached.

NOTE

1. This function supports only permanent and unlogged tables. Temporary tables are not supported.
2. ANALYZE is not automatically triggered twice within 10 seconds for the same table.

enable_invisible_indexes

Parameter description: Specifies whether the optimizer can use invisible indexes.

NOTE

After an index is set to invisible, the performance of query statements may be affected. If you do not want to change the index visibility status and want to use invisible indexes, set **enable_invisible_indexes** to **on**.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: The optimizer can use invisible indexes.
- **off**: The optimizer cannot use invisible indexes.

Default value: **off**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

enable_dynamic_samplesize

Parameter description: Specifies whether to dynamically adjust the number of sampled rows. For a large table with more than one million rows, the number of sampled rows is dynamically adjusted during statistics collection to improve statistics accuracy.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: indicates that the function is enabled.
- **off**: indicates that the function is disabled.

Default value: **on**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

NOTE

The function of dynamically adjusting the number of sampled rows supports only absolute sampling.

STATS_HISTORY_RECORD_LIMIT

Parameter description: Specifies the maximum number of historical statistics that can be retained for each object (including tables, columns, partitions, and indexes). When collecting statistics about an object, the system saves the statistics to the historical statistics table. When the number of statistics about the object in the historical statistics table reaches the threshold and new statistics are collected, the earlier statistics are deleted.

Parameter type: integer.

Unit: none

Value range: 0 to 100

Default value: **10**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: The default value is recommended. If you need to record statistics of more historical versions, increase the value of this parameter. However, the performance of ANALYZE may be affected.

STATS_HISTORY_RETENTION_TIME

Parameter description: Specifies the retention period of historical statistics about each object (including tables, columns, partitions, and indexes). When collecting statistics about an object, the system saves the statistics to the historical statistics table. If the retention period of the statistics about the object in the historical statistics table exceeds the threshold, the system deletes the statistics that exceed the retention period when collecting new statistics.

Parameter type: floating point

Unit: day

Value range: **–1** or a value ranging from 0 to 365000. **–1** indicates that the history statistics are not cleared over time.

Default value: **31**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: The default value is recommended. If you need to record statistics of earlier versions, increase the value of this parameter. However, the performance of ANALYZE may be affected.

default_statistic_granularity

Parameter description: Specifies which partition-level statistics of a partitioned table are collected by default when **PARTITION MODE** is not specified. This parameter does not take effect for non-partitioned tables.

Parameter type: enumerated type

Unit: none

Value range: enumerated values

- **ALL**: collects statistics about the entire table, level-1 partitions, and level-2 partitions.
- **GLOBAL**: indicates that the statistics of the entire table are collected.
- **PARTITION**: indicates that statistics of the level-1 partition are collected.
- **GLOBAL_AND_PARTITION**: indicates that statistics about the entire table and level-1 partitions are collected.
- **SUBPARTITION**: collects statistics about level-2 partitions.
- **ALL_COMPLETE**: collects statistics about the entire table, level-1 partitions, and level-2 partitions.

Default value: **ALL**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. If partition-level statistics need to be collected, you can set this parameter as required. However, the **ANALYZE** performance may be affected.

14.3.9 SPM

SPM_ENABLE_PLAN_CAPTURE

Parameter description: Specifies the SPM plan capture mode.

Parameter type: enumerated type

Unit: none

Value range: **off**, **auto**, **manual**, and **store**

- **off**: indicates that the plan capture function is disabled.
- **auto**: indicates that the automatic plan capture function is enabled. In this mode, the prerequisite for capturing an SQL plan is that the SQL plan is executed twice or more.
- **manual**: indicates that the manual plan capture function is enabled. In this mode, the prerequisite for capturing an SQL plan does not need to meet the requirement that the SQL plan is executed twice or more.

- **store**: The status of all plans is captured as UNACC based on the behavior of the **auto** option.

Default value: **off**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none

NOTICE

During the upgrade, the outline of the new plan is not captured regardless of whether this parameter is enabled. During the upgrade, the upgrade may be rolled back. If a new outline is stored, the outline version compatibility issue may occur after the upgrade rollback.

SPM_ENABLE_PLAN_SELECTION

Parameter description: Specifies whether to enable the SPM plan selection function.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: indicates that the plan selection function is enabled.
- **off**: indicates that the plan selection function is disabled.

Default value: **off**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none

SPM_PLAN_CAPTURE_FILTER

Parameter description: Specifies the database and schema to be captured.

Parameter type: string

Unit: none

Value range: A maximum of 10,000 schemas can be captured, including duplicate schemas.

NOTE

The parameter format is "$db1_oid:$schema1_oid, $db2_oid:$schema2_oid..", for example, "33245:56432,44321:12332,55432:65432", in which:

- **33245:56432** indicates that the SQL plan whose DB OID is **33245** and schema OID is **56432** is to be captured.
- **44321:12332** indicates that the SQL plan whose DB OID is **44321** and schema OID is **12332** is to be captured.
- **55432:65432** indicates that the SQL plan whose DB OID is **55432** and schema OID is **65432** is to be captured.

Default value: an empty string, indicating that all schemas are captured during SPM plan capture and plan selection.

Setting method: This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none

SPM_PLAN_GLOBAL_CACHED_SIZE

Parameter description: Specifies the size of the SPM global plan cache.

Parameter type: integer

Unit: KB

Value range: 10240 to 20971520

Default value: **1048576**

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: You are advised to set this parameter for database nodes in heavy-load scenarios.

SPM_PLAN_SESSION_CACHED_SIZE

Parameter description: Specifies the size of the SPM session plan cache.

Parameter type: integer

Unit: KB

Value range: 1 to 1048576

Default value: 1024

Setting method: This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: You are advised to set this parameter for database nodes in heavy-load scenarios.

SPM_PLAN_CAPTURE_MAX_PLANNUM

Parameter description: Specifies the maximum number of flush plans for a single database.

Parameter type: integer

Unit: none

Value range: 1 to 1000000

Default value: **10000**

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none

SPM_PLAN_RETENTION_DAYS

Parameter description: Specifies the retention period of unused plans on disks.

Parameter type: integer.

Unit: day

Value range: 1 to 1095

Default value: **365**

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none

SPM_ENABLE_LOG_DETAIL

Parameter description: Specifies whether to enable the notice SPM key execution process when the SPM is used. This parameter is usually used together with the following two parameters to view the SPM execution process in gsql: **client_min_messages=notice;logging_module='on(SPM_KEY_FLOW)'**.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: enabled.
- **off**: disabled.

Default value: **off**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none

SPM_ENABLE_BASELINE_CLEANUP

Parameter description: Specifies whether to clear baseline data periodically.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: The periodic baseline clearance function is enabled.
- **off**: The periodic baseline clearance function is disabled.

Default value: **on**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none

14.3.10 Error Reporting and Logging

14.3.10.1 Logging Destination

log_destination

Parameter description: GaussDB supports several methods of logging server messages. Set this parameter to a list of desired log destinations separated by commas. (For example, **log_destination** can be set to **"stderr,csvlog"**.)

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string.

The valid values are **stderr**, **csvlog**, and **syslog**.

- **stderr** indicates that logs are printed to the screen.
- **csvlog** indicates that logs are output in comma separated value (CSV) format. The prerequisite for generating logs in CSV format is that **logging_collector** must be set to **on**. For details, see **Using CSV Log Output**.
- **syslog** indicates that logs are recorded using the syslog of the OS. GaussDB can record logs using syslog from **LOCAL0** to **LOCAL7**. For details, see **syslog_facility**. To record logs using syslog, add the following information to syslog daemon's configuration file:
  ```
  local0.* /var/log/omm
  ```

Default value: stderr

logging_collector

Parameter description: Specifies whether to enable the logger thread to collect logs. This thread captures log messages sent to **stderr** or **csvlog** and redirects them into log files.

This method is more effective than recording logs to syslog because some types of messages cannot be displayed in syslog output, such as messages indicating the loading failures of dynamic link libraries and error messages generated by scripts.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

It is possible to log to **stderr** without using the parameter **logging_collector** and the log messages will go to where the server's **stderr** is directed. However, this method is only suitable for low log volumes due to difficulties in rotating log files.

Value range: Boolean

- **on** indicates that the log collection is enabled.
- **off** indicates that the log collection is disabled.

Default value: **on**

log_directory

Parameter description: Specifies the directory for storing log files when **logging_collector** is set to **on**. The value can be an absolute path, or relative to the data directory. The **log_directory** parameter can be dynamically modified using the **gs_guc reload** command. Only the sysadmin user can access this parameter.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

- If **log_directory** in the configuration file is set to an invalid path, the database cannot be started.
- If you modify the **log_directory** parameter using the **gs_guc reload** command, and the specified path is valid, the log files are output to this new path. If the specified path is invalid, the log files are output to the valid path set last time and the database operation is not affected. The invalid value of **log_directory** is still written into the configuration file.
- In the sandbox environment, the path cannot contain **/var/chroot**. For example, if the absolute path of log is **/var/chroot/var/lib/log/Ruby/gs_log/cn_log**, you only need to set the path to **/var/lib/log/Ruby/gs_log/cn_log**.

NOTE

- Valid path: Users have read and write permissions on the path.
- Invalid path: Users do not have read or write permission on the path.

Value range: a string.

Default value: specified during installation.

log_filename

Parameter description: Specifies the names of generated log files when **logging_collector** is set to **on**. The value is treated as a strftime pattern, so %-escapes can be used to specify time-varying file names. Only the sysadmin user can access this parameter.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

- You are advised to use %-escapes to specify the log file names for efficient management of log files.
- If **log_destination** is set to **csvlog**, log files are output in CSV format with timestamped names, for example, **server_log.1093827753.csv**.

Value range: a string.

Default value: **gaussdb-***%Y-%m-%d_%H%M%S*.**log**

log_file_mode

Parameter description: Specifies the permissions of log files when **logging_collector** is set to **on**. The value of **log_file_mode** is usually a number in the format acceptable to the **chmod** and **umask** system calls.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

- Before setting this parameter, set **log_directory** to store the logs to a directory other than the data directory.
- Do not make the log files world-readable because they might contain sensitive data.

Value range: an octal integer ranging from 0000 to 0777 (that is, 0 to 511 in the decimal format).

NOTE

- **0600** indicates that log files are readable and writable only to the server administrator.
- **0640** indicates that log files are readable and writable to members of the administrator's group.

Default value: 0600

log_truncate_on_rotation

Parameter description: Specifies the writing mode of the log files when **logging_collector** is set to **on**.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Example:

To keep 7 days of logs, one log file per day named **server_log.Mon**, **server_log.Tue**, etc., and automatically overwrite last week's log with this week's log, set **log_filename** to **server_log.%a**, **log_truncate_on_rotation** to **on**, and **log_rotation_age** to **1440**, indicating that the valid duration of the log file is one day.

Value range: Boolean

- **on** indicates that GaussDB overwrites the existing log files of the same name on the server.
- **off** indicates that GaussDB appends the logging messages to the existing log files of the same name on the server.

Default value: **off**

log_rotation_age

Parameter description: Specifies the interval for creating a log file when **logging_collector** is set to **on**. If the duration from the time when the last log file was created to the current time is greater than the value of **log_rotation_age**, a new log file will be generated.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 35791394. The unit is min. **0** indicates that the time-based creation of new log files is disabled.

Default value: **1440**

log_rotation_size

Parameter description: Specifies the maximum size of a server log file when **logging_collector** is set to **on**. If the total size of messages in a log file exceeds the specified value, a log file will be generated.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to *INT_MAX*/1024. The unit is KB.

0 indicates that the capacity-based creation of new log files is disabled.

It is recommended that the unit of the value be MB or bigger, so that log files can be of proper size.

Default value: 20 MB

syslog_facility

Parameter description: Specifies the syslog facility to be used when **log_destination** is set to **syslog**.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: enumerated values. Valid values are **local0**, **local1**, **local2**, **local3**, **local4**, **local5**, **local6**, and **local7**.

Default value: local0

syslog_ident

Parameter description: Specifies the identifier of GaussDB messages in syslog logs when **log_destination** is set to **syslog**.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string.

Default value: **postgres**

event_source

Parameter description: This parameter takes effect only in a Windows environment and is not supported in GaussDB. It specifies the identifier of GaussDB messages in logs when **log_destination** is set to **eventlog**.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string.

Default value: **PostgreSQL**

14.3.10.2 Logging Time

client_min_messages

Parameter description: Specifies which level of messages will be sent to the client. Each level covers all the levels following it. The lower the level is, the fewer messages are sent.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

A same value for **client_min_messages** and **log_min_messages** does not indicate the same level.

Value range: enumerated values. Valid values are **debug**, **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, **info**, **log**, **notice**, **warning**, **error**, **fatal**, and **panic**. For details about the parameters, see **Table 14-8**. If the configured level is higher than **error**, for example, **fatal** or **panic**, the system changes the level to **error** by default.

Default value: **notice**

log_min_messages

Parameter description: Specifies which level of messages will be written into the server log. Each level covers all the levels following it. The lower the level is, the fewer messages will be written into the log.

NOTICE

A same value for **client_min_messages** and **log_min_messages** does not indicate the same level. For some log information, after this parameter is enabled, you also need to set **logging_module** to enable log printing for the corresponding module.

Value type: enumerated type

Unit: none

Value range: **debug**, **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, **info**, **log**, **notice**, **warning**, **error**, **fatal**, and **panic**. For details about the parameters, see **Table 14-8**.

Default value: **warning**

Setting method: This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

log_min_error_statement

Parameter description: Controls which SQL statements that cause an error condition are recorded in the server log.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: enumerated values. Valid values are **debug**, **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, **info**, **log**, **notice**, **warning**, **error**, **fatal**, and **panic**. For details about the parameters, see **Table 14-8**.

NOTE

- The default is **error**, indicating that statements causing errors, log messages, fatal errors, or panics will be logged.
- **panic** indicates that this feature is disabled.

Default value: **error**

log_min_duration_statement

Parameter description: Specifies the threshold for logging the duration of a completed statement. The duration of each completed statement is logged if the statement ran for at least the specified number of milliseconds.

Setting **log_min_duration_statement** makes it easy to trace query statements that need to be optimized. For clients using extended query protocols, the time required for parsing, binding, and executing steps are logged independently.

Parameter type: integer

Unit: ms

NOTICE

When you use this option together with **log_statement**, the text of statements that have been logged by **log_statement** will not be repeatedly logged. If you are not using **syslog**, it is recommended that you log the process ID (PID) or session ID using **log_line_prefix** so that you can link the statement message to the latest duration message.

Value range: –1 to 2147483647.

- If this parameter is set to **250**, all SQL statements that run for 250 ms or longer will be logged.
- **0** indicates that the execution durations of all the statements are logged.
- **–1** indicates that the duration logging is disabled.

Default value: **3000ms** (that is, 3s)

Setting method: This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

backtrace_min_messages

Parameter description: Prints the function's stack information to the server's log file if the information generated is greater than or equal to the level specified by this parameter.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

This parameter is used to locate problems on-site. Frequent stack printing will affect the system's overhead and stability. Therefore, you are advised not to set **backtrace_min_messages** to a level other than fatal and panic when locating problems.

Value range: enumerated values

Valid values are **debug**, **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, **info**, **log**, **notice**, **warning**, **error**, **fatal**, and **panic**. For details about the parameters, see **Table 14-8**.

Default value: **panic**

Table 14-8 explains message severities used by GaussDB. If logging output is sent to **syslog** or **eventlog**, the severities are translated as shown in the table. (Note that the translation takes effect only in a Windows environment where GaussDB does not involve this parameter.)

Table 14-8 Message severity levels

Severity	Description	System Log	Event Log
debug[1-5]	Provides detailed debug information.	DEBUG	INFORMATION
log	Reports information of interest to administrators, for example, checkpoint activity.	INFO	INFORMATION

Severity	Description	System Log	Event Log
info	Provides information implicitly requested by users, for example, output from **VACUUM VERBOSE**.	INFO	INFORMATION
notice	Provides information that might be helpful to users, for example, truncation of long identifiers and index created as part of the primary key.	NOTICE	INFORMATION
warning	Provides warnings of likely problems, for example, **COMMIT** outside a transaction block.	NOTICE	WARNING
error	Reports an error that causes a command to terminate.	WARNING	ERROR
fatal	Reports the reason that causes a session to terminate.	ERR	ERROR
panic	Reports an error that caused all database sessions to terminate.	CRIT	ERROR

14.3.10.3 Logging Content

debug_print_parse

Parameter description: Specifies whether to print parsing tree results.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the printing is enabled.
- **off** indicates that the printing is disabled.

Default value: **off**

debug_print_rewritten

Parameter description: Specifies whether to print query rewriting results.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the printing is enabled.
- **off** indicates that the printing is disabled.

Default value: **off**

debug_print_plan

Parameter description: Specifies whether to print the query execution plan to logs.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the printing is enabled.
- **off** indicates that the printing is disabled.

Default value: **off**

NOTICE

- Debugging information about **debug_print_parse**, **debug_print_rewritten**, and **debug_print_plan** is printed only when the log level is set to **log** or higher. When these parameters are set to **on**, their debugging information will be recorded in server logs and will not be sent to client logs. You can change the log level by setting **client_min_messages** and **log_min_messages**.
- Do not call the gs_encrypt_aes128 and gs_decrypt_aes128 functions when **debug_print_plan** is set to **on**, preventing the risk of sensitive information disclosure. You are advised to filter parameter information of the gs_encrypt_aes128 and gs_decrypt_aes128 functions in the log files generated when **debug_print_plan** is set to **on** before providing the log files to external maintenance engineers for fault locating. After you finish using the logs, delete them as soon as possible.

debug_pretty_print

Parameter description: Indents the logs produced by **debug_print_parse**, **debug_print_rewritten**, and **debug_print_plan**. The output format is more readable but much longer than that generated when this parameter is set to **off**.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the indentation is enabled.
- **off** indicates that the indentation is disabled.

Default value: **on**

log_checkpoints

Parameter description: Specifies whether the statistics on checkpoints and restart points are recorded in the server logs. When this parameter is set to **on**, statistics on checkpoints and restart points are recorded in the log messages, including the number of buffers written and the time spent in writing them.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the statistics on checkpoints and restart points are recorded in the server logs.
- **off** indicates that the statistics on checkpoints and restart points are not recorded in the server logs

Default value: **off**

log_connections

Parameter description: Specifies whether to record logs of connection requests from the client. The log information includes the IP address, port number, username, database name, and duration of key steps in the GaussDB database connection setup process.

Parameter type: Boolean.

Unit: none

NOTE

Some client programs (such as gsql) attempt to connect to the database twice when determining whether a password is required. Therefore, the log message may contain duplicate "connection receive" (indicating that a connection request is received).

Value range:

- **on** indicates that the logs are recorded.
- **off** indicates that the logs are not recorded.

Default value: **off**

Setting method: This is a BACKEND parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

log_disconnections

Parameter description: Specifies whether to record disconnection request information of the client.

This is a BACKEND parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the request information is recorded.
- **off** indicates that the request information is not recorded.

Default value: **off**

log_duration

Parameter description: Specifies whether to record the duration of every completed SQL statement. For clients using extended query protocols, the time required for parsing, binding, and executing steps are logged independently.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **off**: Compared with this option, **log_min_duration_statement** forcibly records the query text.
- If this parameter is set to **on** and **log_min_duration_statement** is set to a positive value, the duration of each completed statement is logged but the query text is included only for statements exceeding the threshold. This behavior can be used for gathering statistics in high-load situation.

Default value: **off**

log_error_verbosity

Parameter description: Specifies the content output to the server log for each message that is logged.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: enumerated values

- **terse** indicates that the output excludes the DETAIL, HINT, QUERY, and CONTEXT error information.
- **verbose** indicates that the output includes the SQLSTATE error code, the source code file name, function name, and number of the line in which the error occurs.
- **default** indicates that the output includes the DETAIL, HINT, QUERY, and CONTEXT error information, and excludes the SQLSTATE error code, the source code file name, function name, and number of the line in which the error occurs.

Default value: **default**

log_hostname

Parameter description: By default, connection log messages only show the IP address of the connecting host. The host name can be recorded when this parameter is set to **on**. It may take some time to parse the host name. Therefore, the database performance may be affected.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the host name is simultaneously recorded.
- **off** indicates that the host name is not simultaneously recorded.

Default value: **off**

log_line_prefix

Parameter description: Specifies the prefix format of each log information. A prefix is a printf-style string that is output at the beginning of each line of the log. The "escape sequences" which begin with % are replaced with status information as listed in **Table 14-9**.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Table 14-9 Escape characters

Escape Character	Effect
%a	Application name
%u	Username
%d	Database name
%r	Remote host name or IP address and remote port. If **log_hostname** is set to **off**, only the IP address and remote port are displayed.
%h	Remote host name or IP address. If **log_hostname** is set to **off**, only the IP address is displayed.
%p	Thread ID.
%t	Timestamp without milliseconds.
%m	Timestamp with milliseconds.
%n	Node from which an error is reported.
%i	Command tag: type of command executed in the current session.
%e	SQLSTATE error code.
%c	Session ID. For details, see the note below the table.
%l	Number of the log line for each session or thread, starting at 1
%s	Thread startup time.
%v	Virtual transaction ID (backend ID/local XID).
%x	Transaction ID (**0** indicates that no transaction ID is assigned).
%q	Produces no output. If the current thread is a backend thread, this escape sequence is ignored and subsequent escape sequences are processed. Otherwise, this escape sequence and subsequent escape sequences are all ignored.
%S	Session ID.
%T	Trace ID
%%	The character %.

NOTE

The %c escape character prints a session ID consisting of two 4-byte hexadecimal numbers separated by a period (.). The numbers are the thread startup time and the thread ID. Therefore, %c can also be used as a space saving way of printing those items. For example, execute the following query to generate the session ID from pg_stat_activity:

```
SELECT to_hex(EXTRACT(EPOCH FROM backend_start)::integer) || '.' ||
       to_hex(pid)
FROM pg_stat_activity;
```

- If you set a nonempty value for **log_line_prefix**, you should usually make its last character be a space, to provide visual separation from the rest of the log line. A punctuation character can be used, too.
- Syslog generates its own timestamp and thread ID information. Therefore, you do not need to include those escape characters when you are logging in to syslog.

Value range: a string.

Default value: **%m %n %u %d %h %p %S %x %a**

NOTE

%m %n %u %d %h %p %S %x %a indicates the session start timestamp, error reporting node, username, database name, remote host name or IP address, thread ID, session ID, transaction ID, and application name.

log_lock_waits

Parameter description: If the time for which a session waits to acquire a lock is longer than the value of **deadlock_timeout**, this parameter specifies whether to record this message in the database. This is useful in determining if lock waits are causing poor performance.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the information is recorded.
- **off** indicates that the information is not recorded.

Default value: **off**

log_statement

Parameter description: Specifies which SQL statements are recorded. For clients using extended query protocols, logging occurs when an execute message and values (enclosed by a pair of single quotation marks) of bound parameters are received.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

- Statements that contain simple syntax errors are not logged even if **log_statement** is set to **all**, because logging occurs only after basic parsing has been completed and the statement type is determined. If an extended query protocol is used, statements that fail before the execution phase (during parsing or planning) are not logged, either. Set **log_min_error_statement** to **ERROR** or lower to log such statements.
- If this parameter is set to a value other than **none**, the statement audit function is enabled. The database administrator can access server logs to view SQL execution records.

Value range: enumerated values

- **none** indicates that no statement is recorded.
- **ddl** indicates that all data definition statements, such as CREATE, ALTER, and DROP, are recorded.
- **mod** indicates that all DDL statements and data modification statements, such as INSERT, UPDATE, DELETE, TRUNCATE, and COPY FROM, are recorded.
- **all** indicates that all statements, including the PREPARE, EXECUTE, and EXPLAIN ANALYZE statements, are recorded.

Default value: **none**

log_temp_files

Parameter description: Specifies the deletion information about temporary files that meet recording requirements. Temporary files can be created for sorting, hashing, and storing temporary querying results. If the recording is enabled, a log entry is generated for each temporary file when it is deleted.

Parameter type: integer.

Unit: KB

Value range: –1 to 2147483647.

- A positive value indicates that the deletion information of temporary files whose size is larger than the value specified by **log_temp_files** is recorded.
- **0** indicates that the delete information of all temporary files is recorded.
- **–1** indicates that the delete information of any temporary files is not recorded.

Default value: **–1**

Setting method: This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value **–1** which indicates that no temporary file deletion information is recorded. If this parameter is set, the more temporary files that meet the recording requirements, the more logs are recorded, which affects the system performance.

log_timezone

Parameter description: Specifies the time zone used for timestamps written in the server log. Different from **TimeZone**, this parameter takes effect for all sessions in the database.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. You can query the PG_TIMEZONE_NAMES view to obtain the value. For details, see "System Catalogs and System Views > System Views > PG_TIMEZONE_NAMES" in *Developer Guide*.

Default value: Set this parameter based on the OS time zone.

NOTE

The default value will be changed when **gs_initdb** is used to set system environments.

logging_module

Parameter description: Specifies whether module logs are output on the server. This is a session-level parameter, and you are advised not to use the gs_guc tool to set it.

Parameter type: string.

Unit: none

Default value: empty

Value range: Logs of this module are generated on the server. Logs of other modules are not generated on the server, but can be viewed by running **SHOW logging_module**.

```
ALL,on(),off(COMMAND,DFS,GUC,GSCLEAN,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,OBS,INDEX,EXECUTOR,OPFUSION,GPC,GSC,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_CHOICE,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANNER,OPT_STAT_EXT,SPM,SPM_KEY_FLOW,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PLANHINT,PARQUET,PGSTAT,CARBONDATA,SNAPSHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,WDR_SNAPSHOT,WDR_REPORT,ASP_REPORT,INCRE_CKPT,INCRE_BG_WRITER,DBL_WRT,RTO_RPO,HEARTBEAT,COMM_IPC,COMM_PARAM,TIMESERIES,SCHEMA,GTT,SEGMENT_PAGE,LIGHTPROXY,HOTKEY,THREAD_POOL,OPT_AI,WALRECEIVER,USTORE,UPAGE,UBTREE,UNDO,TIMECAPSULE,GEN_COL,DCF,AI4DB,DB4AI,ABO,MOD_ABOFEEDBACK,PLDEBUGGER,ADVISOR,SEC,SEC_FE,SEC_LEGER,SEC_POLICY,SEC_SDD,SEC_TDE,COMM_FRAMEWORK,COMM_PROXY,COMM_POOLER,COMM_STATUS,VACUUM,JOB,SPI,NEST_COMPILE,RESOWNER,GSSTACK,LOGICAL_DECODE,GPRC,DISASTER_READ,STANDBY_READ,REPSYNC,SQLPATCH,PARTITION,UBT_NEWPAGE,GPI,GS_DEPENDENCY,LWLOCK,LOCK,UNIQUE_SQL,GLC,SRF,DBLINK,BARRIER_CREATOR,EXRTO_PAGE_P,SEQ_TUP_P,BT_TUP_P,DISPATCH_VERIFY,HBKT,DBE_STATS,DBE_XMLGEN,GS_ILM,GSPERF,EXEC_REMOTE,UBTREE_PARA,BTREE_PARA,ANTI_CACHE,ANTI_RECYCLER,VERIFYLOG,GS_REPAIR,AUTHID,CCINDEX)
```

CAUTION

CN_RETRY does not take effect in the current version.

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting method: Run **SHOW logging_module** to view which modules are controllable. For example, the query output result is as follows:


```
gaussdb=# show logging_module;
logging_module
-----------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------
ALL,on(),off(COMMAND,DFS,GUC,GSCLEAN,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBL
SPC,WLM,OBS,INDEX,EXECUTOR,OPFUSION,GPC,GSC,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT
_JOIN,OPT_AGG,OPT_CHOICE,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANNER,OPT_STAT_EXT,SPM,SP
M_KEY_FLOW,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PLANHINT,PARQUET,PGSTAT,CARBONDATA,SNAPS
HOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,W
DR_SNAPSHOT,WDR_REPORT,ASP_REPORT,INCRE_CKPT,INCRE_BG_WRITER,DBL_WRT,RTO_RPO,HEARTBEAT,
COMM_IPC,COMM_PARAM,TIMESERIES,SCHEMA,GTT,SEGMENT_PAGE,LIGHTPROXY,HOTKEY,THREAD_POOL,
OPT_AI,WALRECEIVER,USTORE,UPAGE,UBTREE,UNDO,TIMECAPSULE,GEN_COL,DCF,AI4DB,DB4AI,ABO,MOD_
ABOFEEDBACK,PLDEBUGGER,ADVISOR,SEC,SEC_FE,SEC_LEGER,SEC_POLICY,SEC_SDD,SEC_TDE,COMM_FRAM
EWORK,COMM_PROXY,COMM_POOLER,COMM_STATUS,VACUUM,JOB,SPI,NEST_COMPILE,RESOWNER,GSST
ACK,LOGICAL_DECODE,GPRC,DISASTER_READ,STANDBY_READ,REPSYNC,SQLPATCH,PARTITION,UBT_NEWPA
GE,GPI,GS_DEPENDENCY,LWLOCK,LOCK,UNIQUE_SQL,GLC,SRF,DBLINK,BARRIER_CREATOR,EXRTO_PAGE_P,SE
Q_TUP_P,BT_TUP_P,DISPATCH_VERIFY,HBKT,DBE_STATS,DBE_XMLGEN,GS_ILM,GSPERF,EXEC_REMOTE,UBTREE
_PARA,BTREE_PARA,ANTI_CACHE,ANTI_RECYCLER,VERIFYLOG,GS_REPAIR,AUTHID,CCINDEX)
(1 row)
```

Controllable modules are identified by uppercase letters, and the special ID **ALL** is used for setting all module logs. You can control the output of module logs by setting the parameter to **on** or **off**. Enable log output for SSL:

```
gaussdb=# set logging_module='on(SSL)';
SET
gaussdb=# show
logging_module;
        logging_module
-----------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------
------------------------------------------------
ALL,on(SSL),off(COMMAND,DFS,GUC,GSCLEAN,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,
TBLSPC,WLM,OBS,INDEX,EXECUTOR,OPFUSION,GPC,GSC,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,
OPT_JOIN,OPT_AGG,OPT_CHOICE,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANNER,OPT_STAT_EXT,SPM,
SPM_KEY_FLOW,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PLANHINT,PARQUET,PGSTAT,CARBONDATA,SNAP
SHOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,
WDR_SNAPSHOT,WDR_REPORT,ASP_REPORT,INCRE_CKPT,INCRE_BG_WRITER,DBL_WRT,RTO_RPO,HEARTBEA
T,COMM_IPC,COMM_PARAM,TIMESERIES,SCHEMA,GTT,SEGMENT_PAGE,LIGHTPROXY,HOTKEY,THREAD_POO
L,OPT_AI,WALRECEIVER,USTORE,UPAGE,UBTREE,UNDO,TIMECAPSULE,GEN_COL,DCF,AI4DB,DB4AI,ABO,MOD
_ABOFEEDBACK,PLDEBUGGER,ADVISOR,SEC,SEC_FE,SEC_LEGER,SEC_POLICY,SEC_SDD,SEC_TDE,COMM_FRA
MEWORK,COMM_PROXY,COMM_POOLER,COMM_STATUS,VACUUM,JOB,SPI,NEST_COMPILE,RESOWNER,GS
STACK,LOGICAL_DECODE,GPRC,DISASTER_READ,STANDBY_READ,REPSYNC,SQLPATCH,PARTITION,UBT_NEW
PAGE,GPI,GS_DEPENDENCY,LWLOCK,LOCK,UNIQUE_SQL,GLC,SRF,DBLINK,BARRIER_CREATOR,EXRTO_PAGE_P
,SEQ_TUP_P,BT_TUP_P,DISPATCH_VERIFY,HBKT,DBE_STATS,DBE_XMLGEN,GS_ILM,GSPERF,EXEC_REMOTE,UBT
REE_PARA,BTREE_PARA,ANTI_CACHE,ANTI_RECYCLER,VERIFYLOG,GS_REPAIR,AUTHID,CCINDEX)
(1 row)
```

SSL log output is enabled.

The **ALL** identifier can be used to quickly enable or disable log output for all modules.

```
gaussdb=# set logging_module='off(ALL)';
SET
gaussdb=# show
logging_module;
          logging_module
-----------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------
ALL,on(),off(COMMAND,DFS,GUC,GSCLEAN,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBL
SPC,WLM,OBS,INDEX,EXECUTOR,OPFUSION,GPC,GSC,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT
_JOIN,OPT_AGG,OPT_CHOICE,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANNER,OPT_STAT_EXT,SPM,SP
M_KEY_FLOW,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PLANHINT,PARQUET,PGSTAT,CARBONDATA,SNAPS
HOT,XACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,W
DR_SNAPSHOT,WDR_REPORT,ASP_REPORT,INCRE_CKPT,INCRE_BG_WRITER,DBL_WRT,RTO_RPO,HEARTBEAT,
COMM_IPC,COMM_PARAM,TIMESERIES,SCHEMA,GTT,SEGMENT_PAGE,LIGHTPROXY,HOTKEY,THREAD_POOL,
```

```
OPT_AI,WALRECEIVER,USTORE,UPAGE,UBTREE,UNDO,TIMECAPSULE,GEN_COL,DCF,AI4DB,DB4AI,ABO,MOD_
ABOFEEDBACK,PLDEBUGGER,ADVISOR,SEC,SEC_FE,SEC_LEGER,SEC_POLICY,SEC_SDD,SEC_TDE,COMM_FRAM
EWORK,COMM_PROXY,COMM_POOLER,COMM_STATUS,VACUUM,JOB,SPI,NEST_COMPILE,RESOWNER,GSST
ACK,LOGICAL_DECODE,GPRC,DISASTER_READ,STANDBY_READ,REPSYNC,SQLPATCH,PARTITION,UBT_NEWPA
GE,GPI,GS_DEPENDENCY,LWLOCK,LOCK,UNIQUE_SQL,GLC,SRF,DBLINK,BARRIER_CREATOR,EXRTO_PAGE_P,SE
Q_TUP_P,BT_TUP_P,DISPATCH_VERIFY,HBKT,DBE_STATS,DBE_XMLGEN,GS_ILM,GSPERF,EXEC_REMOTE,UBTREE
_PARA,BTREE_PARA,ANTI_CACHE,ANTI_RECYCLER,VERIFYLOG,GS_REPAIR,AUTHID,CCINDEX)
(1 row)

gaussdb=# set logging_module='on(ALL)';
SET
gaussdb=# show
logging_module;
        logging_module
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------
ALL,on(COMMAND,DFS,GUC,GSCLEAN,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,CACHE,ADIO,SSL,GDS,TBLSPC,
WLM,OBS,INDEX,EXECUTOR,OPFUSION,GPC,GSC,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOI
N,OPT_AGG,OPT_CHOICE,OPT_SUBPLAN,OPT_SETOP,OPT_SKEW,OPT_PLANNER,OPT_STAT_EXT,SPM,SPM_KE
Y_FLOW,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PLANHINT,PARQUET,PGSTAT,CARBONDATA,SNAPSHOT,X
ACT,HANDLE,CLOG,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,REDO,FUNCTION,PARSER,INSTR,WDR_S
NAPSHOT,WDR_REPORT,ASP_REPORT,INCRE_CKPT,INCRE_BG_WRITER,DBL_WRT,RTO_RPO,HEARTBEAT,COM
M_IPC,COMM_PARAM,TIMESERIES,SCHEMA,GTT,SEGMENT_PAGE,LIGHTPROXY,HOTKEY,THREAD_POOL,OPT_
AI,WALRECEIVER,USTORE,UPAGE,UBTREE,UNDO,TIMECAPSULE,GEN_COL,DCF,AI4DB,DB4AI,ABO,MOD_ABOF
EEDBACK,PLDEBUGGER,ADVISOR,SEC,SEC_FE,SEC_LEGER,SEC_POLICY,SEC_SDD,SEC_TDE,COMM_FRAMEWO
RK,COMM_PROXY,COMM_POOLER,COMM_STATUS,VACUUM,JOB,SPI,NEST_COMPILE,RESOWNER,GSSTACK,L
OGICAL_DECODE,GPRC,DISASTER_READ,STANDBY_READ,REPSYNC,SQLPATCH,PARTITION,UBT_NEWPAGE,GP
I,GS_DEPENDENCY,LWLOCK,LOCK,UNIQUE_SQL,GLC,SRF,DBLINK,BARRIER_CREATOR,EXRTO_PAGE_P,SEQ_TU
P_P,BT_TUP_P,DISPATCH_VERIFY,HBKT,DBE_STATS,DBE_XMLGEN,GS_ILM,GSPERF,EXEC_REMOTE,UBTREE_PAR
A,BTREE_PARA,ANTI_CACHE,ANTI_RECYCLER,VERIFYLOG,GS_REPAIR,AUTHID,CCINDEX),off()
(1 row)
```

Dependency: The value of this parameter depends on the settings of **log_min_messages**.

opfusion_debug_mode

Parameter description: Checks whether simple queries are optimized for debugging. If this parameter is set to **log**, you can view the specific reasons why queries are not optimized in the database node execution plans.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: enumerated values

- **off** indicates that reasons why queries are not optimized are not included.
- **log** indicates that reasons why queries are not optimized are included in the database node execution plan.

NOTICE

To view the reasons why queries are not optimized in the log, set the parameter to **log**, **log_min_messages** to **debug4**, and **logging_module** to **on(OPFUSION)**. Note that a large number of log messages may be generated. Therefore, execute only a small number of jobs during debugging.

Default value: **off**

enable_debug_vacuum

Parameter description: Specifies whether to allow output of some VACUUM-related logs for problem locating. This parameter is used only by developers. Common users are advised not to use it.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**/**true**: enabled.
- **off**/**false**: disabled.

Default value: **off**

14.3.10.4 Using CSV Log Output

Prerequisites

- The **log_destination** parameter is set to **csvlog**.
- The **logging_collector** parameter is set to **on**.

Definition of csvlog

Log lines are emitted in comma separated values (CSV) format.

An example table definition for storing CSV-format log output is shown as follows:

```
CREATE TABLE gaussdb_log
(
log_time timestamp(3) with time zone,
node_name text,
user_name text,
database_name text,
process_id bigint,
connection_from text,
"session_id" text,
session_line_num bigint,
command_tag text,
session_start_time timestamp with time zone,
virtual_transaction_id text,
transaction_id bigint,
query_id bigint,
module text,
error_severity text,
sql_state_code text,
message text,
detail text,
hint text,
internal_query text,
internal_query_pos integer,
context text,
query text,
query_pos integer,
location text,
application_name text
);
```

For details, see **Table 14-10**.

Table 14-10 Meaning of each csvlog field

Column	Description	Column	Description
log_time	Timestamp in milliseconds	module	Module to which the log belongs
node_name	Node name	error_severity	ERRORSTATE code
user_name	Username	sql_state_code	SQLSTATE code
database_name	Database name	message	Error message
process_id	Thread ID	detail	Detailed error message
connection_from	Port number of the client host	hint	Prompt message
session_id	Session ID	internal_query	Internal query (This field is used to query the information leading to errors if any.)
session_line_num	Number of lines in each session	internal_query_pos	Pointer for an internal query
command_tag	Command tag	context	Environment
session_start_time	Start time of a session	query	Character count at the position where errors occur
virtual_transaction_id	Regular transaction	query_pos	Pointer at the position where errors occur
transaction_id	Transaction ID	location	Position where errors occur in the GaussDB source code if **log_error_verbosity** is set to **verbose**
query_id	Query ID	application_name	Application name

Run the **COPY FROM** command to import a log file to this table.

```
COPY gaussdb_log FROM '/opt/data/gs_log/logfile.csv' WITH csv;
```


NOTE

The log name (**logfile.csv**) here needs to be replaced with the name of a log generated.

Simplifying Input

Simplify importing CSV log files by performing the following operations:

- Set **log_filename** and **log_rotation_age** to provide a consistent, predictable naming solution for log files. By doing this, you can predict when an individual log file is complete and ready to be imported.
- Set **log_rotation_size** to **0** to disable size-based log rollback, as it makes the log file name difficult to predict.
- Set **log_truncate_on_rotation** to **on** so that old log data cannot be mixed with the new one in the same file.

14.3.11 Alarm Detection

During the running of a database, error scenarios can be detected so that users are informed of the errors in time. You can view system_alarm logs in *$GAUSSLOG*/**cm**, *$GAUSSLOG*/**gs_log**, or *$GAUSSLOG*/**roach/agent**.

enable_alarm

Parameter description: Specifies whether to enable the alarm detector thread to detect fault scenarios that may occur in the database.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the alarm detector thread is enabled.
- **off** indicates that the alarm detector thread is disabled.

Default value: **on**

NOTE

This parameter takes effect only on DNs.

connection_alarm_rate

Parameter description: Specifies the ratio restriction on the maximum number of allowed parallel connections to the database. The maximum number of concurrent connections to the database is **max_connections** x **connection_alarm_rate**.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a floating-point number ranging from 0.0 to 1.0

Default value: **0.9**

alarm_report_interval

Parameter description: Specifies the interval at which an alarm is reported.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer. The unit is s.

Default value: **10**

alarm_component

Parameter description: Certain alarms are suppressed during alarm reporting. That is, the same alarm will not be repeatedly reported by an instance within the period specified by **alarm_report_interval**. Its default value is **10s**. In this case, the parameter specifies the location of the alarm component that is used to process alarm information. Only the sysadmin user can access this parameter.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string.

- If **--alarm-type** in the gs_preinstall script is set to **5**, no third-party component is connected and alarms are written into system_alarm logs. In this case, the value of **alarm_component** is **/opt/huawei/snas/bin/snas_cm_cmd**.
- If **--alarm-type** in the gs_preinstall script is set to **1**, a third-party component is connected. In this case, the value of **alarm_component** is the absolute path of the executable program of the third-party component.

Default value: **/opt/huawei/snas/bin/snas_cm_cmd**

table_skewness_warning_threshold

Parameter description: Specifies the threshold for triggering a table skew alarm.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a floating-point number ranging from 0 to 1

Default value: **1**

table_skewness_warning_rows

Parameter description: Specifies the minimum number of rows for triggering a table skew alarm.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to *INT_MAX*

Default value: **100000**

14.3.12 Statistics During the Database Running

14.3.12.1 Query and Index Statistics Collector

The query and index statistics collector is used to collect statistics during database running. The statistics include the times of inserting and updating a table and index, the number of disk blocks and tuples, and the time required for the last cleanup and analysis on each table. The statistics can be viewed by querying system views pg_stats and pg_statistic. The following parameters are used to set the statistics collection feature in the server scope:

track_activities

Parameter description: Collects statistics about the commands that are being executed in each session. For a stored procedure, if this parameter is enabled, you can view the PERFORM statement, stored procedure calling statement, SQL statement, and OPEN CURSOR statement that are being executed in the stored procedure in the pg_stat_activity view.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the statistics collection function is disabled.

Default value: **on**

⚠ CAUTION

If this parameter is set to **off**, the memory recycling capability of the storage engine is affected, causing space expansion.

---

track_counts

Parameter description: Collects statistics about database activities.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the statistics collection function is disabled.

NOTE

Database statistics are required when the autovacuum process checks for databases that need to be vacuumed. Therefore, the default value is set to **on**.

Default value: **on**

⚠ CAUTION

If this parameter is set to **off**, the memory recycling capability of the storage engine is affected, causing space expansion.

---

track_procedure_sql

Parameter description: Specifies whether the SQL statements that are being executed in the stored procedure are printed in the query column in the pg_stat_activity system catalog.

Parameter type: Boolean

Unit: none

Value range: on or **off**

- **on**: indicates that when a stored procedure is called, the statements that are being executed by the stored procedure are printed in the query column of **pg_stat_activity**.
- **off**: indicates that when a stored procedure is called, only statements for invoking the stored procedure are printed in the query column of **pg_stat_activity**.

Default value: **on**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none

track_io_timing

Parameter description: Collects statistics about I/O timing in the database. The I/O timing statistics can be queried by using the **pg_stat_database** parameter.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- If this parameter is set to **on**, the collection function is enabled. In this case, the collector repeatedly queries the OS at the current time. As a result, large number of costs may occur on some platforms. Therefore, the default value is set to **off**.
- **off** indicates that the statistics collection function is disabled.

Default value: **off**

track_functions

Parameter description: Collects statistics of the number and duration of function invocations.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

When the SQL functions are set to inline functions queried by the invoking, these SQL functions cannot be traced no matter these functions are set or not.

Value range: enumerated values

- **pl** indicates that only procedural language functions are traced.
- **all** indicates that SQL language functions area traced.
- **none** indicates that the function tracing function is disabled.

Default value: **none**

track_activity_query_size

Parameter description: Specifies byte counts of the current running commands used to trace each active session. If the actual number of bytes in a command is greater than the value of this parameter, the command is truncated.

Parameter type: integer

Unit: byte

Value range: 100 to 102400

Default value: 1024

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Set this parameter based on the actual service scenario.

update_process_title

Parameter description: Collects statistics updated with a process name each time the server receives a new SQL statement.

The process name can be viewed by running the **ps** command.

This is an INTERNAL parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the statistics collection function is disabled.

Default value: **off**

stats_temp_directory

Parameter description: Specifies the directory for storing temporary statistics. Only the sysadmin user can access this parameter.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

If a RAM-based file system directory is used, the actual I/O cost can be lowered and the performance can be improved.

Value range: a string

Default value: **pg_stat_tmp**

track_thread_wait_status_interval

Parameter description: Specifies the interval of collecting the thread status information.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: 0 to 1 day. The unit is min.

Default value: **30min**

enable_save_datachanged_timestamp

Parameter description: Specifies whether to record the time when **INSERT**, **UPDATE**, **DELETE**, or **EXCHANGE**/**TRUNCATE**/**DROP PARTITION** is performed on table data.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the time when an operation is performed on table data will be recorded.
- **off** indicates that the time when an operation is performed on table data will not be recorded.

Default value: **on**

enable_plan_trace

Parameter description: Specifies whether to enable the PLAN TRACE feature for the database. This parameter cannot be set globally by running the **gs_guc** command. It can be set only by running the **set** command in a connected session.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the PLAN TRACE feature is enabled.
- **off** indicates that the PLAN TRACE feature is disabled.

Default value: **off**

plan_collect_thresh

Parameter description: Collects statistics about the plans that are being executed in each session.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from –1 to 2147483647

- **–1**: Plans that are being executed are not collected.
- **0**: Plans that are being executed are collected before plan execution.

- A value greater than **0** indicates that when the total number of tuples incrementally returned by all operators in a plan is greater than or equal to the value of this parameter, plans that are being executed are collected once.

Default value: **0**

track_sql_count

Parameter description: Collects statistics about the statements (SELECT, INSERT, UPDATE, MERGE INTO, and DELETE) that are being executed in a session.

In the x86-based centralized deployment scenario, the hardware configuration specifications are 32-core CPU and 256 GB memory. When the Benchmark SQL 5.0 tool is used to test performance, the performance fluctuates by about 0.8% by enabling or disabling this parameter.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the statistics collection function is disabled.

Default value: **on**

NOTE

- The **track_sql_count** parameter is restricted by the **track_activities** parameter.
  - If **track_activities** is set to **on** and **track_sql_count** is set to **off**, a warning message indicating that **track_sql_count** is disabled will be displayed in logs when the **gs_sql_count** view is queried.
  - If both **track_activities** and **track_sql_count** are set to **off**, two warning messages indicating that **track_activities** is disabled and **track_sql_count** is disabled will be displayed in logs when the views are queried.
  - If **track_activities** is set to **off** and **track_sql_count** is set to **on**, a warning message indicating that **track_activities** is disabled will be displayed in logs when the views are queried.
- If this parameter is disabled, the query result is 0.

14.3.13 Autovacuum

The autovacuum thread automatically runs the **VACUUM** and **ANALYZE** statements to recycle the record space marked as deleted and update statistics about the table.

The autovacuum thread contains the VACUUM and ANALYZE processes. The **autovacuum_naptime** parameter specifies the interval between them. The default interval is 10 minutes. You can set the interval based on the actual service scenario. The interval cannot be completely accurate. It depends on the hardware conditions and load of the environment. If a large amount of data needs to be vacuumed, the heavy load may delay the ANALYZE process. It also depends on the values of **autovacuum_naptime** and **autovacuum_max_workers**. **autovacuum_naptime** specifies the execution interval. A smaller value indicates a shorter execution interval. However, data vacuuming and statistics calculation involved increase the CPU usage, memory usage, and I/O overhead. **autovacuum_max_workers** specifies the maximum number of concurrent

automatic cleanup threads. A larger value makes the parameter more compatible with **autovacuum_naptime**, but the CPU usage, memory usage, and I/O overhead also increase.

In addition, you can set the **immediate_analyze_threshold** parameter to trigger ANALYZE when the amount of new data exceeds the threshold. For details about how to calculate the threshold, see the description of **immediate_analyze_threshold**.

autovacuum

Parameter description: Specifies whether to start the autovacuum thread in the database. Ensure that the **track_counts** parameter is set to **on** before starting the autovacuum thread.

NOTE

- Set the **autovacuum** parameter to **on** to automatically vacuum two-phase transactions after the system recovers from faults.
- If **autovacuum** is set to **on** and **autovacuum_max_workers** to **0**, the autovacuum process is started only when the system recovers from faults to clean up abnormal two-phase transactions.
- If **autovacuum** is set to **on** and the value of **autovacuum_max_workers** is greater than **0**, the system will automatically vacuum the two-phase transactions and processes after recovering from faults.

Parameter type: Boolean.

Unit: none

Value range:

- **on** indicates that the autovacuum thread is started.
- **off** indicates that the autovacuum thread is not started.

Default value: **on**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

autovacuum_mode

Parameter description: Specifies whether the autoanalyze and autovacuum function are enabled. This parameter is valid only when **autovacuum** is set to **on**.

Value type: enumerated type

Unit: none

Value range: **analyze**, **vacuum**, **mix**, and **none**

- **analyze** indicates that only autoanalyze is performed.
- **vacuum** indicates that only autovacuum is performed.
- **mix** indicates that both autoanalyze and autovacuum are performed.
- **none** indicates that neither autoanalyze nor autovacuum is performed.

Default value: **mix**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

autoanalyze_timeout

Parameter description: Specifies the timeout interval of autoanalyze. If the duration of autoanalyze on a table exceeds the value of **autoanalyze_timeout**, the autoanalyze operation is automatically canceled.

Parameter type: integer.

Unit: second

Value range: 0 to 2147483. The value **0** indicates that no timeout occurs.

Default value: **300**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none.

autovacuum_io_limits

Parameter description: Specifies the upper limit of I/Os triggered by the autovacuum thread per second.

Parameter type: integer.

Unit: none

Value range: –1 to 1073741823. **–1** indicates that the default Cgroup is used.

Default value: **–1**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

log_autovacuum_min_duration

Parameter description: Records each step performed by the autovacuum process to the server log when the execution time of the autovacuum process is greater than or equal to a certain value. This parameter helps track the autovacuum behavior.

A setting example is as follows:

Set the **log_autovacuum_min_duration** parameter to 250 ms to record the actions of autovacuum if it runs for 250 ms or longer.

Value range: an integer ranging from –1 to 2147483647. The unit is ms.

- **0** indicates that all autovacuum actions are recorded in the log.
- **–1** indicates that all autovacuum actions are not recorded in the log.
- A value other than **–1** indicates that a message is recorded when an autovacuum action is skipped due to a lock conflict. (The reason for skipping autovacuum is recorded for audit.)

Default value: **–1**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: The default value is **–1**. If the autovacuum operation needs to be recorded in logs, set this parameter to a value other than **–1**.

autovacuum_max_workers

Parameter description: Specifies the maximum number of autovacuum worker threads that can run at the same time. The upper limit of this parameter is related to the values of **max_connections** and **job_queue_processes**.

Parameter type: integer.

Unit: none

Value range: an integer. The minimum value is **0**, indicating that autovacuum is not enabled. The theoretical maximum value is **262143**, but the actual maximum value is a dynamic value calculated by the following formula: 262143 – **max_inner_tool_connections** – **max_connections job_queue_processes** – **max_concurrent_autonomous_transactions** – Number of auxiliary threads – Number of autovacuum launcher threads – 1. The number of auxiliary threads and the number of autovacuum launcher threads are specified by two macros. Their default values are **20** and **2** respectively.

Default value: **3**

Adjustment suggestion: If this parameter is set to a larger value, more autovacuum processes are created and more CPU and memory resources are occupied. Therefore, you are advised not to set this parameter to a large value. Otherwise, memory cannot be allocated or too many CPU resources are occupied, causing database startup errors or affecting services.

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

autovacuum_naptime

Parameter description: Specifies the interval between activity rounds for the autovacuum process.

Parameter type: integer.

Unit: second

Value range: 1 to 2147483.

Default value: **10min** (that is, 600s)

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**. For example, if the value is **600** without a unit, **autovacuum_naptime** indicates 600s. If the value is **10min**, **autovacuum_naptime** indicates 10 minutes. If the unit is required, the value must be **s**, **min**, **h**, or **d**.

Setting suggestion: Retain the default value.

autovacuum_vacuum_threshold

Parameter description: Used to calculate the threshold for triggering VACUUM. The formula is as follows: **autovacuum_vacuum_threshold** + **reltuples** (number of tuples in a table) x **autovacuum_vacuum_scale_factor**. When the number of deleted or updated records in a table exceeds the threshold, the VACUUM operation is executed on this table.

Parameter type: integer.

Unit: none

Value range: 0 to 2147483647

Default value: **50**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

autovacuum_analyze_threshold

Parameter description: Used to calculate the threshold for triggering the ANALYZE operation. The formula is as follows: **autovacuum_analyze_threshold** + **reltuples** (number of tuples in a table) x **autovacuum_analyze_scale_factor**. When the number of deleted, inserted, or updated records in a table exceeds the threshold, the ANALYZE operation is executed on this table.

Parameter type: integer.

Unit: none

Value range: 0 to 2147483647

Default value: **50**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

autovacuum_vacuum_scale_factor

Parameter description: The scale factor for vacuuming a table, which is used to calculate the threshold for triggering VACUUM. The formula is as follows: **autovacuum_vacuum_threshold** + **reltuples** (number of tuples in a table) x **autovacuum_vacuum_scale_factor**. When the number of deleted or updated records in a table exceeds the threshold, the VACUUM operation is executed on this table.

Parameter type: floating-point.

Unit: none

Value range: 0.0 to 100.0

Default value: **0.2**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: A larger value makes it harder to trigger VACUUM, which has less impact on performance.

autovacuum_analyze_scale_factor

Parameter description: Specifies the scale factor for analyzing a table, which is used to calculate the threshold for executing ANALYZE. The formula is as follows: **autovacuum_analyze_threshold** + **reltuples** (number of tuples in the table) x **autovacuum_analyze_scale_factor**. When the number of deleted, inserted, or updated records in a table exceeds the threshold, the ANALYZE operation is executed on this table.

Parameter type: floating-point.

Unit: none

Value range: 0.0 to 100.0

Default value: **0.1**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: A larger value makes it harder to trigger ANALYZE, which has less impact on performance.

autovacuum_freeze_max_age

Parameter description: Specifies the maximum age (in transactions) that an Astore table's **pg_class.relfrozenxid** field can retain before a VACUUM operation is forcibly performed.

- The old files under the subdirectory of **pg_clog/** can also be deleted by the VACUUM operation.
- Even if the autovacuum thread is not started, the system will call the thread.

Parameter type: long integer

Unit: none

Value range: 100000 to 576460752303423487

Default value: **4000000000**

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

autovacuum_vacuum_cost_delay

Parameter description: Specifies the value of the cost delay used in the autovacuum operation.

Parameter type: integer.

Unit: ms

Value range: –1 to 100. **–1** indicates that the regular vacuum cost delay is used.

Default value: 20

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

autovacuum_vacuum_cost_limit

Parameter description: Sets the value of the cost limit used in the autovacuum operation.

Parameter type: integer.

Unit: none

Value range: –1 to 10000. **–1** indicates that the regular vacuum cost limit is used.

Default value: **–1**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

defer_csn_cleanup_time

Parameter description: Specifies the local recycling interval.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 2147483647. The unit is ms.

Default value: **5s** (5000 ms)

14.3.14 Default Settings of Client Connection

14.3.14.1 Statement Behavior

This section describes related default parameters involved in the execution of SQL statements.

search_path

Parameter description: Specifies the order in which schemas are searched when an object is referenced with no schema specified. The value of this parameter consists of one or more schema names. Different schema names are separated by commas (,).

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

- If schemas of temporary tables exist in the current session, the schemas can be listed in the search path by using the alias **pg_temp**, for example, **'pg_temp,public'**. The schema of temporary tables has the highest search priority and is always searched before all the other schemas specified in **pg_catalog** and **search_path**. Therefore, do not explicitly specify **pg_temp** to be searched after other schemas in **search_path**. This setting will not take effect and an error message will be displayed. If the alias **pg_temp** is used,

the temporary schema will be searched only for tables, views, and data types, and not for functions or operators.

- The system catalog schema, pg_catalog, has the second highest search priority and is the first to be searched among all the schemas, excluding **pg_temp**, specified in **search_path**. Therefore, do not explicitly specify **pg_catalog** to be searched after other schemas in **search_path**. This setting will not take effect and an error message will be displayed.
- When an object is created without a specific target schema, the object will be placed in the first valid schema listed in **search_path**. An error is reported if the search path is empty.
- The current effective value of the search path can be examined through the SQL function current_schema. This is different from examining the value of **search_path**, because the current_schema function displays the first valid schema name in **search_path**.

Value range: a string

NOTE

- When this parameter is set to **"$user", public**, shared use of a database (where no users have private schemas, and all share use of public), private per-user schemas and combinations of them are supported. Other effects can be obtained by modifying the default search path setting, either globally or per-user.
- When this parameter is set to a null string (''), the system automatically converts it into a pair of double quotation marks ("").
- If the content contains double quotation marks, the system considers them as insecure characters and converts each double quotation mark into a pair of double quotation marks.

Default value: **"$user",public**

NOTE

$user indicates the name of the schema with the same name as the current session user. If the schema does not exist, **$user** will be ignored.

current_schema

Parameter description: Specifies the current schema.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string

Default value: **"$user",public**

NOTE

$user indicates the name of the schema with the same name as the current session user. If the schema does not exist, **$user** will be ignored.

If you need to obtain a schema during kernel development, use the value of **search_path** because the schema is determined by **search_path**. For compatibility, **current_schema** is used only to modify the value of **search_path**.

default_tablespace

Parameter description: Specifies the default tablespace of the created objects (tables and indexes) when a **CREATE** command does not explicitly specify a tablespace.

- The value of this parameter is either the name of a tablespace, or an empty string that indicates the use of the default tablespace of the current database. If a non-default tablespace is specified, users must have CREATE privilege for it. Otherwise, creation attempts will fail.
- This parameter is not used for temporary tables. For them, the **temp_tablespaces** is used instead.
- This parameter is not used when users create databases. By default, a new database inherits its tablespace setting from the template database.

  This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that the default tablespace is used.

Default value: empty

temp_tablespaces

Parameter description: Specifies one or more tablespaces to which temporary objects (temporary tables and their indexes) will be created when a CREATE command does not explicitly specify a tablespace. Temporary files for sorting large data sets are created in these tablespaces.

The value of this parameter can be a list of names of tablespaces. When there is more than one name in the list, GaussDB chooses a random tablespace from the list upon the creation of a temporary object each time. However, within a transaction, successively created temporary objects are placed in successive tablespaces in the list. If the element selected from the list is an empty string, GaussDB will automatically use the default tablespace of the current database instead.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. An empty string indicates that all temporary objects are created only in the default tablespace of the current database. For details, see **default_tablespace**.

Default value: empty

check_function_bodies

Parameter description: Specifies whether to enable validation of the function body string during the execution of **CREATE FUNCTION**. Verification is occasionally disabled to avoid problems, such as forward references when you restore function definitions from a dump. After the function is enabled, the word syntax of the PL/SQL in the stored procedure is verified, including the data type, statement, and expression. The SQL statements in the stored procedure are not checked in the CREATE phase. Instead, they are checked during running.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that validation of the function body string is enabled during the execution of **CREATE FUNCTION**.
- **off** indicates that validation of the function body string is disabled during the execution of **CREATE FUNCTION**.

Default value: **on**

default_transaction_isolation

Parameter description: Specifies the default isolation level of each transaction.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

NOTE

The current version does not support the setting of the default transaction isolation level. The default value is **read committed**. Do not change the value.

Value range: enumerated values

- **read committed** indicates that the data read by a transaction is committed at the moment it is read.
- **repeatable read** indicates that the data that has been read by the current transaction cannot be modified by other transactions until the current transaction completes, thereby preventing unrepeatable reads.
- **serializable**: Currently, this isolation level is not supported in GaussDB. It is equivalent to **repeatable read**.

Default value: read committed

default_transaction_read_only

Parameter description: Specifies whether each new transaction is in read-only state.

CAUTION

If this parameter is set to **on**, the DML and write transactions cannot be executed.

Parameter type: Boolean

Unit: none

Value range:

- **on** indicates that the transaction is in read-only state.
- **off** indicates that the transaction is not in read-only state.

Default value: **off**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

default_transaction_deferrable

Parameter description: Specifies the default deferrable status of each new transaction. It currently has no effect on read-only transactions or those running at isolation levels lower than serializable.

GaussDB does not support the serializable isolation level. Therefore, the parameter takes no effect.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that a transaction is delayed by default.
- **off** indicates that a transaction is not delayed by default.

Default value: **off**

session_replication_role

Parameter description: Specifies the behavior of replication-related triggers and rules for the current session.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

Setting this parameter will discard all the cached execution plans.

Value range: enumerated values

- **origin** indicates that the system copies operations such as insert, delete, and update from the current session.
- **replica** indicates that the system copies operations such as insert, delete, and update from other places to the current session.
- **local** indicates that the system will detect the role that has logged in to the database when using the function to copy operations and will perform related operations.

Default value: **origin**

statement_timeout

Parameter description: If the statement execution time (starting from the time the server receives the command) is longer than the duration specified by the parameter, error information is displayed and the statement exits.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**. The default value is **0**, indicating that the parameter does not take effect.

Value range: an integer ranging from 0 to 2147483647. The unit is ms.

Default value: **0**

vacuum_freeze_min_age

Parameter description: Specifies whether VACUUM replaces the **xmin** column of a record with **FrozenXID** when scanning a table (in the same transaction).

Parameter type: integer

Unit: none

Value range: 0 to 576460752303423487

NOTE

Although you can set this parameter to any value, VACUUM will limit the effective value to 50% of **autovacuum_freeze_max_age** by default.

Default value: **2000000000**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

vacuum_freeze_table_age

Parameter description: Specifies when VACUUM scans the whole table and freezes old tuples. VACUUM performs a full table scan if the difference between the current transaction ID and the value of **pg_class.relfrozenxid64** is greater than the specified time.

Parameter type: integer

Unit: none

Value range: 0 to 576460752303423487

NOTE

Although you can set this parameter to any value, **VACUUM** will limit the effective value to 95% of **autovacuum_freeze_max_age** by default. Therefore, a periodic manual **VACUUM** has a chance to run before an anti-wraparound autovacuum is launched for the table.

Default value: **4000000000**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

bytea_output

Parameter description: Specifies the output format for values of the bytea type.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: enumerated values

- **hex** indicates the binary data is converted to the two hexadecimal digits per byte.
- **escape** uses ASCII character sequences to represent binary strings, and converts those binary strings that cannot be represented as ASCII characters into special escape sequences.

Default value: **hex**

xmlbinary

Parameter description: Specifies how binary values are to be encoded in XML.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: enumerated values

- base64
- hex

Default value: **base64**

xmloption

Parameter description: Specifies whether DOCUMENT or CONTENT is implicit when converting between XML and string values.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: enumerated values

- **document** indicates an HTML document.
- **content** indicates a common string.

Default value: **content**

max_compile_functions

Parameter description: Specifies the maximum number of function compilation results stored in the server.

Parameter type: integer

Unit: none

Value range: 1 to 2147483647.

Default value: **1000** (196-core CPU/1536 GB memory, 128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 64-core CPU/512 GB memory, 60-core CPU/480 GB memory, 32-core CPU/256 GB memory, 16-core CPU/128 GB memory, 8-core CPU/64 GB memory, 4-core CPU/32 GB memory); **10** (4-core CPU/16 GB memory)

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Excessive functions and compilation results of stored procedures may occupy large memory space. Setting this parameter to a proper value can reduce the memory usage and improve system performance.

14.3.14.2 Locale and Formatting

This section describes parameters related to the time format setting.

DateStyle

Parameter description: Specifies the display format for date and time values, as well as the rules for interpreting ambiguous date input values.

This variable contains two independent components: the output format declaration (ISO, Postgres, SQL, or German) and the input/output order of year/month/day (DMY, MDY, YMD, Euro, European, US, NonEuro, NonEuropean, or Default). The two components can be set separately or together. The keywords Euro and European are synonyms for DMY. The keywords US, NonEuro, and NonEuropean are synonyms for MDY.

Parameter type: string

Unit: none

Value range: combination of the output format declaration and the input/output order of year/month/day. (The two components can be set separately.)

- Output format declaration: ISO, Postgres, SQL, or German
- Input/Output order of year/month/day: DMY (Euro, European), MDY (US, NonEuro, NonEuropean), or YMD

Default value: **"ISO, MDY"**

NOTE

gs_initdb will initialize this parameter so that its value is the same as that of **lc_time**.

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: The ISO format is recommended.

IntervalStyle

Parameter description: Specifies the display format for interval values.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: enumerated values

- **sql_standard** indicates that output matching SQL standards will be generated.
- **postgres** indicates that output matching PostgreSQL 8.4 will be generated when the **DateStyle** parameter is set to **ISO**.
- **postgres_verbose** indicates that output matching PostgreSQL 8.4 will be generated when the **DateStyle** parameter is set to **non_ISO**.
- **iso_8601** indicates that output matching the time interval "format with designators" defined in ISO 8601 will be generated.
- **a** indicates the output that matches the numtodsinterval function. For details, see "SQL Reference > Functions and Operators > Date and Time Processing Functions and Operators > numtodsinterval" in *Developer Guide*.

NOTICE

The **IntervalStyle** parameter also affects the interpretation of ambiguous interval input.

Default value: **postgres**

TimeZone

Parameter description: Specifies the time zone for displaying and interpreting timestamps.

Parameter type: string

Unit: none

Value range: You can query the PG_TIMEZONE_NAMES view to obtain the value. For details, see "System Catalogs and System Views > System Views > PG_TIMEZONE_NAMES" in *Developer Guide*.

Default value: **"PRC"**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

NOTE

gs_initdb will set a time zone value that is consistent with the system environment.

timezone_abbreviations

Parameter description: Specifies the time zone abbreviations that will be accepted by the server.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. **India**, **Australia**, or **Default**.

Default value: **Default**

NOTE

Default indicates an abbreviation that works in most of the world, which is applicable to most cases. There are also other abbreviations, such as **'Australia'** and **'India'** that can be defined for a particular installation. For other time zone abbreviations, you need to set them in the corresponding configuration files before creating the database.

extra_float_digits

Parameter description: Adjusts the number of digits displayed for floating-point values, including float4, float8, and geometric data types. The parameter value is added to the standard number of digits (FLT_DIG or DBL_DIG as appropriate).

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from –15 to 3.

NOTE

- This parameter can be set to **3** to include partially-significant digits. It is especially useful for dumping float data that needs to be restored exactly.
- This parameter can also be set to a negative value to suppress unwanted digits.

Default value: **0**

client_encoding

Parameter description: Specifies the client-side encoding (character set).

Set this parameter based on the situation of the front-end services. Try to keep the encoding consistent on the client and server to improve efficiency.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: encoding compatible with PostgreSQL. **UTF8** indicates that the database encoding is used.

NOTE

- You can run the **locale -a** command to check the system-supported locales and the corresponding encodings, and select one as required. The sorting rules may vary according to OSs or versions of the same OS. When application services are logically migrated between these OSs, differences and changes in OS sorting rules may cause differences in database functions such as indexes, partitions, and sorting operators. For example, data result sets returned by a same range query statement are different, or query results returned by a same sorting query statement are different. The application service needs to check whether the service data contains related characters based on the locale differences released by the OS.
- By default, **gs_initdb** will initialize the setting of this parameter based on the current system environment. You can also run the **locale** command to check the current configuration environment.
- To use consistent encoding for communication within the database, you are advised to retain the default value of **client_encoding**. Modification to this parameter in the **gaussdb.conf** file (by using the gs_guc tool, for example) does not take effect.
- **client_encoding** can be set to **GB18030_2022**. When **client_encoding** is set to **gb18030_2022**, ensure that the GB18030 character set supported by the client OS has been upgraded to version 2022.
- If the encoding format of the database character set is UTF8, the encoding format of the region supported by the current system is GB18030, the GB18030 character set supported by the client OS has been upgraded to version 2022, and the database contains historical data stored before the GB18030 character set is upgraded, the following situations may occur:
  1. When **client_encoding** is set to **gb18030**, the character encoding of historical data in the UTF8 database returned to the client is the same as that before the client character set is upgraded. However, the font of 38 characters whose mapping changes during the character set upgrade is the same as that in version 2022.
  2. When **client_encoding** is set to **gb18030_2022**, the character encoding of historical data in the UTF8 database returned to the client are the same as that after the upgrade of the client character set and the font of characters are the same as that before the upgrade. This is because the versions of the GB18030 character set are not completely compatible, which may cause data inconsistency. Therefore, if the GB18030 character set needs to be upgraded on the client and historical data exists, you need to convert the historical data before the upgrade.
- **client_encoding** can be set to **ZHS16GBK**. The following table lists the setting methods of **server_encoding** and **client_encoding** corresponding to all ways that ZHS16GBK, GB18030, and GB18030_2022 character sets can be transformed.

server_encoding	client_encoding	locale	Setting Method
zhs16gbk	utf8	utf8	The database automatically obtains the value of **locale**.
utf8	zhs16gbk	gbk	• Users use gsql to manually set **client_encoding** to **zhs16gbk**. • When JDBC is used for connection, the URL parameter is used to set **characterEncoding** to **zhs16gbk**.
zhs16gbk	zhs16gbk	gbk	• When gsql is used for connection, the database automatically obtains the value of **locale**. After internal processing, **client_encoding** is automatically set to **zhs16gbk**. • When JDBC is used for connection, the URL parameter is used to set **characterEncoding** to **zhs16gbk**.

zhs16gbk	gb18030	gb18030	• When gsql is used for connection, the database automatically obtains the value of **locale**. • When JDBC is used for connection, the URL parameter is used to set **characterEncoding** to **gb18030**.
gb18030	zhs16gbk	gbk	• gsql does not support this setting. • When JDBC is used for connection, the URL parameter is used to set **characterEncoding** to **zhs16gbk**.
zhs16gbk	gb18030-2022	gb18030	• When gsql is used for connection, users can manually set **client_encoding** to **gb18030_2022**. • When JDBC is used for connection, the URL parameter is used to set **characterEncoding** to **gb18030_2022**.

gb18030-2022	zhs16gbk	gbk	• gsql does not support this setting. • When JDBC is used for connection, the URL parameter is used to set **characterEncoding** to **zhs16gbk**.
gb18030	utf8	utf8	The database automatically obtains the value of **locale**.
utf8	gb18030	gb18030	• The database automatically obtains the value of **locale**. • When JDBC is used for connection, the URL parameter is used to set **characterEncoding** to **gb18030**.
gb18030_2022	utf8	utf8	The database automatically obtains the value of **locale**.
utf8	gb18030_2022	gb18030	• When gsql is used for connection, users can manually set **client_encoding** to **gb18030_2022**. • When JDBC is used for connection, the URL parameter is used to set **characterEncoding** to **gb18030_2022**.

gb18030	gb18030	gb18030	• The database automatically obtains the value of **locale**. • When JDBC is used for connection, the URL parameter is used to set **characterEncoding** to **gb18030**.
gb18030_2022	gb18030_2022	gb18030	• When gsql is used for connection, the database automatically obtains the value of **locale**. After internal processing, **client_encoding** is automatically set to **gb18030_2022**. • When JDBC is used for connection, the URL parameter is used to set **characterEncoding** to **gb18030_2022**.

Default value: **UTF8**

Recommended value: **SQL_ASCII** or **UTF8**

lc_messages

Parameter description: Specifies the language in which messages are displayed.

- Acceptable values are system-related.
- On some systems, this locale category does not exist. Setting this variable will still work, but there will be no effect. In addition, translated messages for the desired language may not exist. In this case, you can still see the English messages.

  This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string

NOTE

- You can run the **locale -a** command to check the system-supported locales and the corresponding encodings, and select one as required.
- By default, **gs_initdb** will initialize the setting of this parameter based on the current system environment. You can also run the **locale** command to check the current configuration environment.

Default value: **C**

lc_monetary

Parameter description: Specifies the display format of monetary values. It affects the output of functions such as **to_char**. Acceptable values are system-related.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string

NOTE

- You can run the **locale -a** command to check the system-supported locales and the corresponding encodings, and select one as required.
- By default, **gs_initdb** will initialize the setting of this parameter based on the current system environment. You can also run the **locale** command to check the current configuration environment.

Default value: **C**

lc_numeric

Parameter description: Specifies the display format of numbers. It affects the output of functions such as **to_char**. Acceptable values are system-related.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string

NOTE

- You can run the **locale -a** command to check the system-supported locales and the corresponding encodings, and select one as required.
- By default, **gs_initdb** will initialize the setting of this parameter based on the current system environment. You can also run the **locale** command to check the current configuration environment.

Default value: **C**

lc_time

Parameter description: Specifies the display format of time and locale. It affects the output of functions such as **to_char**. Acceptable values are system-related.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string

NOTE

- You can run the **locale -a** command to check the system-supported locales and the corresponding encodings, and select one as required.
- By default, **gs_initdb** will initialize the setting of this parameter based on the current system environment. You can also run the **locale** command to check the current configuration environment.

Default value: **C**

lc_time_names

Parameter description: Specifies the language used to display names and abbreviations of dates and months. The output of the DATE_FORMAT(), DAYNAME(), and MONTHNAME() functions is affected. The performance of STR_TO_DATE() and GET_FORMAT() is not affected.

Parameter type: string

Unit: none

Value range: IETF language tags in simple format, such as **en_US** or **zh_CN**.

Default value: **en_US**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Setting GUC Parameters**.

Setting suggestion: Switch the language based on service requirements.

default_week_format

Parameter description: Specifies the default **mode** value of the week() function. The following table lists the value range and description.

default_week_format	First Day of a Week	Range	First Week Definition
0	Sunday	0 to 53	The first week that contains a Sunday in this year.
1	Monday	0 to 53	The first week that contains four or more days in this year.
2	Sunday	1 to 53	The first week that contains a Sunday in this year.
3	Monday	1 to 53	The first week that contains four or more days in this year.
4	Sunday	0 to 53	The first week that contains four or more days in this year.

default_week_format	First Day of a Week	Range	First Week Definition
5	Monday	0 to 53	The first week that contains a Monday in this year.
6	Sunday	1 to 53	The first week that contains four or more days in this year.
7	Monday	1 to 53	The first week that contains a Monday in this year.

NOTE

For week 1 of a year, the number 1 indicates a week number. The week number of the first week of a year may be 0, depending on the value of **week_format**, as shown in the third column.

Parameter type: integer

Unit: none

Value range: [0,7]

Default value: **0**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Setting GUC Parameters**.

Setting suggestion: Select a mode based on service requirements.

14.3.14.3 Other Default Parameters

This section describes the default database loading parameters.

dynamic_library_path

Parameter description: Specifies the path that the system will search for a shared database file that is dynamically loadable. When a dynamically loadable module needs to be opened and the file name specified in the **CREATE FUNCTION** or **LOAD** command does not contain a directory, the system will search this path for the required file. Only the sysadmin user can access this parameter.

The value of **dynamic_library_path** must be a list of absolute paths separated by colons (:). When the name of a path starts with the special variable *$libdir*, the variable will be replaced with the directory in which the module provided by the GaussDB is installed. For example:

```
dynamic_library_path = '/usr/local/lib/gaussdb:/opt/testgs/lib:$libdir'
```

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: string

NOTE

If the value of this parameter is set to an empty character string, the automatic path search is turned off. The dynamic library search paths specified by **local_preload_libraries** and **shared_preload_libraries** also depend on this parameter. If this parameter is left blank or set to an invalid path, the dynamic libraries fail to be loaded, which will result in abnormal database services.

Default value: $libdir

local_preload_libraries

Parameter description: Specifies one or more shared libraries that are to be preloaded at connection start. If multiple libraries are to be loaded, separate their names with commas (,). All library names are converted to lower case unless double-quoted.

- Any user can change this option. Therefore, library files that can be loaded are restricted to those saved in the **plugins** subdirectory of the standard library installation directory. It is the database administrator's responsibility to ensure that libraries in this directory are all safe. Entries in **local_preload_libraries** can specify the library directory explicitly, for example, *$libdir*/**plugins/mylib**, or just specify the library name, for example, **mylib**. (**mylib** is equivalent to *$libdir*/**plugins/mylib**.)
- Unlike **shared_preload_libraries**, there are no differences in performance between loading a module at session start or doing this during the session. The intent of this feature is to allow debugging or performance-measurement libraries to be loaded into specific sessions without an explicit LOAD command. For example, debugging can be enabled under a given username by setting this parameter to **ALTER USER SET**.
- If a specified library is not found, the connection attempt will fail. If the configuration is incorrect (for example, the length exceeds the limit or the directory is invalid), the process cannot be started properly.
- Every GaussDB-supported library has a "magic block" that is checked to guarantee compatibility. For this reason, non-GaussDB-supported libraries cannot be loaded in this way.

This is a BACKEND parameter. Set it based on instructions provided in **Table 14-1**.

Value range: string

Default value: empty

14.3.15 Lock Management

In GaussDB, a deadlock may occur when concurrently executed transactions compete for resources. This section describes parameters used for managing transaction locks.

deadlock_timeout

Parameter description: Specifies the deadlock timeout interval. When the applied lock exceeds the preset value, the system will check whether a deadlock occurs. This parameter takes effect only for common locks.

Parameter type: integer.

Unit: ms

Value range: 1 to 2147483647.

Default value: **1s**

Setting method: This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion:

- The check for deadlock is relatively expensive. Therefore, the server does not check it when waiting for a lock every time. Deadlocks seldom occur when the system is running. Therefore, the system just needs to wait on the lock for a while before checking for deadlocks. Increasing the value of **deadlock_timeout** reduces the time wasted in deadlock check, but slows down reporting of real deadlock errors. On a heavily loaded server, you may need to set **deadlock_timeout** to a larger value. It is recommended that this value exceed the transaction time to avoid deadlock check before locks are released.
- When **log_lock_waits** is set to **on**, **deadlock_timeout** determines a waiting time to write the lock waiting time information during query execution to logs. To study the lock delay, you can set **deadlock_timeout** to a value smaller than the normal value.

lockwait_timeout

Parameter description: Specifies the timeout for attempts to acquire a lock. If the time spent in waiting for a lock exceeds the specified time, an error is reported. This parameter takes effect only for common locks.

Parameter type: integer.

Unit: ms

Value range: 0 to *INT_MAX*

Default value: **20min**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**. For example, if the value is **900** without a unit, **lockwait_timeout** indicates 900 ms. If the value is **20min**, **lockwait_timeout** indicates 20 minutes. If the unit is required, the value must be **ms**, **s**, **min**, **h**, or **d**.

Setting suggestion: Retain the default value. Alternatively, set this parameter based on the service requirements.

update_lockwait_timeout

Parameter description: Specifies the maximum duration that a lock waits for concurrent updates on a row to complete when the concurrent update feature is enabled. If the time spent in waiting for a lock exceeds the specified time, an error is reported. This parameter takes effect only for common locks.

Parameter type: integer.

Unit: ms

Value range: 0 to 2147483647

Default value: **2min** (120000 ms)

Setting method: This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**. For example, if the value is **900** without a unit, **update_lockwait_timeout** indicates 900 ms. If the value is **2min**, **update_lockwait_timeout** indicates 2 minutes. If the unit is required, the value must be **ms**, **s**, **min**, **h**, or **d**.

Setting suggestion: Observe the workload of transactions that concurrently update the same row. For common TP services, the execution time is less than two minutes. In this case, the probability of false positives is low. If a large number of transactions concurrently update the same row and the execution time exceeds two minutes, you can increase the value of this parameter to prevent false positives due to lock timeout.

max_locks_per_transaction

Parameter description: Determines the average number of object locks allocated to each transaction.

Parameter type: integer.

Unit: none

Value range: 10 to 2147483647

Default value:

256 (196-core CPU/1536 GB memory, 128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 64-core CPU/512 GB memory, 60-core CPU/480 GB memory, 32-core CPU/256 GB memory, 16-core CPU/128 GB memory, 8-core CPU/64 GB memory, 4-core CPU/32 GB memory); **64** (4-core CPU/16 GB memory)

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion:

- The maximum number of hash tables that can be locked by a shared lock at any time is calculated based on an assumption: **max_locks_per_transaction** x (**max_connections** + **max_prepared_transactions**) and **max_locks_per_transaction** ≥ Number of concurrent service transactions x Number of object locks added by each service transaction/(**max_connections** + **max_prepared_transactions**). In this case, an upper limit is determined for this parameter.
- Within the specified range, objects can be locked simultaneously at any time. You may need to increase the value of this parameter if many different tables are modified in a single transaction.
- Increasing the value of this parameter may cause GaussDB to request more System V-shared memory than the OS's default configuration allows, leading to database startup failure.

- When running a standby node, you must set this parameter to a value that is no less than that on the primary node. Otherwise, queries will not be allowed on the standby node.

max_pred_locks_per_transaction

Parameter description: Specifies the average number of predicate locks allocated for each transaction.

- The size of the shared predicate lock table is calculated under the condition that a maximum of *N* independent objects need to be locked at any time. *N* = **max_pred_locks_per_transaction** x (**max_connections** + **max_prepared_transactions**). Objects whose amount does not exceed the preset number can be locked simultaneously at any time. You may need to increase this value if many different tables are modified in a single transaction. This parameter can only be set at server start.
- Increasing the value of this parameter may cause GaussDB to request more System V-shared memory than the OS's default configuration allows.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 10 to *INT_MAX*.

Default value: **64**

gs_clean_timeout

Parameter description: Specifies the average interval for clearing temporary tables on the primary node.

- When the database connection is terminated abnormally, temporary tables may exist. In this case, you need to call the gs_clean tool to clear the temporary tables in the database.
- If this parameter is set to a larger value, the time for clearing GaussDB temporary tables may be prolonged.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 2147483. The unit is s.

Default value: **1min**

partition_lock_upgrade_timeout

Parameter description: Specifies the timeout for attempts to upgrade an exclusive lock (read allowed) to an access exclusive lock (read/write blocked) on a partitioned table during the execution of some query statements. If there are concurrent read transactions running, the lock upgrade will need to wait. **partition_lock_upgrade_timeout** sets the waiting timeout for lock upgrade attempts.

- When you do **MERGE PARTITION** and **CLUSTER PARTITION** on a partitioned table, temporary tables are used for data rearrangement and file exchange. To concurrently perform as many operations as possible on the partitions,

exclusive locks are acquired for the partitions during data rearrangement and access exclusive locks are acquired during file exchange.

- Generally, a partition waits until it acquires a lock, or a timeout occurs if the partition waits for a period longer than the value specified by the **lockwait_timeout** parameter.
- When doing **MERGE PARTITION** or **CLUSTER PARTITION** on a partitioned table, an access exclusive lock needs to be acquired during file exchange. If the lock fails to be acquired, the acquisition is retried at an interval of 50 ms until timeout occurs. The **partition_lock_upgrade_timeout** parameter specifies the time to wait before the lock acquisition attempt times out.
- If **partition_lock_upgrade_timeout** is set to **–1**, the lock upgrade never times out. The lock upgrade is continuously retried until it succeeds.

  This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from –1 to 3000. The unit is s.

Default value: **1800**

fault_mon_timeout

Parameter description: Specifies the period for detecting lightweight deadlocks. This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 1440. The unit is minute.

Default value: **5min**

enable_online_ddl_waitlock

Parameter description: Specifies whether to block DDL operations to wait for the release of database locks, such as **pg_advisory_lock**. This parameter is mainly used in online OM operations and you are advised not to modify the settings.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **off**

xloginsert_locks

Parameter description: Specifies the number of locks on concurrent write-ahead logging. This parameter is used to improve the efficiency of writing write-ahead logs.

Parameter type: integer.

Unit: none

Value range: 1 to 1000. If the CPU uses the NUMA architecture, the value must be an integer multiple of the number of NUMA nodes.

Default value: 16

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: When the concurrency pressure of the Arm CPU architecture is high, performance jitter may occur due to atomic operation competition. You can decrease the value of this parameter for stable running performance.

num_internal_lock_partitions

Parameter description: Specifies the number of internal lightweight lock partitions. It is mainly used for performance optimization in various scenarios. The content is organized in the KV format of keywords and numbers. Different types of locks are separated by commas (,). The sequence does not affect the setting result. For example, **CLOG_PART=256,CSNLOG_PART=512** is equivalent to **CSNLOG_PART=512,CLOG_PART=256**. If you set the same keyword multiple times, only the latest setting takes effect. For example, if you set **CLOG_PART** to **256** and **CLOG_PART** to **2**, the value of **CLOG_PART** is **2**. If no keyword is set, the default value is used. The usage description, maximum value, minimum value, and default value of each lock type are as follows:

- **CLOG_PART**: number of Clog file controllers. Increasing the value of this parameter improves the Clog writing efficiency and transaction commit performance, but increases the memory usage. Decreasing the value of this parameter reduces the memory usage, but may increase the conflict of writing Clogs and affect the performance. The value ranges from 1 to 256.
- **CSNLOG_PART**: number of CSNLOG file controllers. Increasing the value of this parameter improves the CSNLOG writing efficiency and transaction commit performance, but increases the memory usage. Decreasing the value of this parameter reduces the memory usage, but may increase the conflict of writing CSNLOGs and affect the performance. The value ranges from 1 to 512.
- **LOG2_LOCKTABLE_PART**: two logarithms of the number of ordinary table lock partitions. Increasing the value can improve the concurrency of obtaining locks in the normal process, but may increase the time required for transferring and clearing locks. When wait events occur in **LockMgrLock**, you can increase the value to improve the performance. The minimum value is 4, that is, the number of lock partitions is 16. The maximum value is 16, that is, the number of lock partitions is 65536.
- **TWOPHASE_PART**: number of partitions of the two-phase transaction lock. Increasing the value can increase the number of concurrent two-phase transaction commits. The value ranges from 1 to 64.
- **FASTPATH_PART**: maximum number of locks that each thread can obtain without using the main lock table. When a partitioned table is read, updated, inserted, or deleted and the wait event is LockMgrLock, you can increase the value of this parameter to prevent LockMgrLock from being obtained and improve performance. It is recommended that the value be greater than or equal to that calculated using the following formula: Number of partitions x (1 + Number of local indexes) + Number of global indexes + 10. Increasing the value will increase the memory usage. The value ranges from 20 to 10000.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string.

Default value:

'CLOG_PART=256,CSNLOG_PART=512,LOG2_LOCKTABLE_PART=4,TWOPHASE_PART=1,FASTPATH_PART=20' (196-core CPU/1536 GB memory, 128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 64-core CPU/512 GB memory, 60-core CPU/480 GB memory, 32-core CPU/256 GB memory, 16-core CPU/128 GB memory, 8-core CPU/64 GB memory, or 4-core CPU/32 GB memory); **'CLOG_PART=8,CSNLOG_PART=16,LOG2_LOCKTABLE_PART=4,TWOPHASE_PART=1,FASTPATH_PART=20'** (4-core CPU/16 GB memory)

enable_wait_exclusive_lock

Parameter description: Specifies whether to enable the hang detection and cure function for the exclusive lock of ProcArrayLock.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **on**

barrier_lock_timeout

Parameter description: Specifies the timeout interval for holding a barrier lock.

Parameter type: integer.

Unit: second

Value range: 0 to 3600

Default value: **30s**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

enable_xid_abort_check

Parameter description: Specifies whether to verify transaction ID rollback is enabled when a transaction is committed.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: enabled.
- **off**: disabled.

Default value: **on**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

14.3.16 Version and Platform Compatibility

14.3.16.1 Compatibility with Earlier Versions

This section describes the parameters that control the backward compatibility and external compatibility of GaussDB. A backward compatible database supports applications of earlier versions. This section describes parameters used for controlling backward compatibility of a database.

array_nulls

Parameter description: Determines whether the array input parser recognizes unquoted NULL as a null array element.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that null values can be entered in arrays.
- **off** indicates backward compatibility with the old behavior. Arrays containing the value **NULL** can still be created when this parameter is set to **off**.

Default value: **on**

backslash_quote

Parameter description: Determines whether a single quotation mark can be represented by \' in a string text.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

When the string text meets the SQL standards, \ has no other meanings. This parameter only affects the handling of non-standard-conforming string texts, including escape string syntax (E'...').

Value range: enumerated values

- **on** indicates that the use of \' is always allowed.
- **off** indicates that the use of \' is rejected.
- **safe_encoding** indicates that the use of \' is allowed only when client encoding does not allow ASCII \ within a multibyte character.

Default value: **safe_encoding**

escape_string_warning

Parameter description: Specifies whether to issue a warning when a backslash (\) is used as an escape in an ordinary character string.

- Applications that wish to use a backslash (\) as an escape need to be modified to use escape string syntax (E'...'). This is because the default behavior of ordinary character strings treats the backslash as an ordinary character in each SQL standard.
- This variable can be enabled to help locate codes that need to be changed.
- If E'...' is used as an escape, logs may be incomplete in some scenarios.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

Default value: **on**

lo_compat_privileges

Parameter description: Specifies whether to enable backward compatibility for the privilege check of large objects.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

on indicates that the privilege check is disabled when large objects are read or modified.

Default value: **off**

quote_all_identifiers

Parameter description: Specifies whether to forcibly quote all identifiers even if they are not keywords when the database generates SQL. This will affect the output of **EXPLAIN** and the results of functions, such as pg_get_viewdef. For details, see the **--quote-all-identifiers** parameter of **gs_dump**.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the forcible quoting is enabled.
- **off** indicates that the forcible quoting is disabled.

Default value: **off**

sql_inheritance

Parameter description: Controls the inheritance semantics. This parameter specifies the access policy of descendant tables. **off** indicates that subtables cannot be accessed by commands. That is, the ONLY keyword is used by default. This setting is compatible with earlier versions.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that subtables can be accessed.
- **off** indicates that subtables cannot be accessed.

Default value: **on**

standard_conforming_strings

Parameter description: Controls whether ordinary string texts ('...') treat backslashes as ordinary texts as specified in the SQL standard.

- Applications can check this parameter to determine how string texts will be processed.
- It is recommended that characters be escaped by using the escape string syntax (E'...').

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the function is enabled.
- **off** indicates that the function is disabled.

Default value: **on**

synchronize_seqscans

Parameter description: Determines sequential scans of tables to synchronize with each other, so that concurrent scans read the same data block at about the same time and share the I/O load.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that a scan may start in the middle of the table and then "wrap around" the end to cover all rows to synchronize with the activity of scans already in progress. This may result in unpredictable changes in the row ordering returned by queries that have no ORDER BY clause.
- **off** indicates that the scan always starts from the table heading.

Default value: **on**

enable_beta_features

Parameter description: Specifies whether to enable some features that are not officially released and are used only for POC verification. Exercise caution when enabling these extended features because they may cause errors in some scenarios.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the features are enabled for forward compatibility. Note that enabling them may cause errors in certain scenarios.
- **off** indicates that the features are disabled.

Default value: **off**

default_with_oids

Parameter description: Specifies whether **CREATE TABLE** and **CREATE TABLE AS** include an **OID** field in newly-created tables if neither **WITH OIDS** nor **WITHOUT OIDS** is specified. It also determines whether OIDs will be included in tables created by **SELECT INTO**.

It is not recommended that OIDs be used in user tables. Therefore, this parameter is set to **off** by default. When OIDs are required for a particular table, **WITH OIDS** needs to be specified during the table creation.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that **CREATE TABLE** and **CREATE TABLE AS** can include an **OID** field in newly-created tables.
- **off** indicates that **CREATE TABLE** and **CREATE TABLE AS** cannot include any OID field in newly-created tables.

Default value: **off**

enable_recordtype_check_strict

Parameter description: Specifies whether to perform strict check on the record type in PL/SQL. For details, see the notice below.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: indicates that the record type created during compilation of stored procedures, functions, and packages is checked for unsupported functions, and the record type created by provided stored procedures and functions is checked for the NOT NULL function.
- **off**: indicates backward compatibility and the record type is not checked.

Default value: **on**

NOTICE

After an upgrade from an earlier version, this parameter is set to **off** by default.

After this parameter is set to **on**, the following three changes occur:

1. For the record type created in a stored procedure or function, the **NOT NULL** column constraint takes effect.

2. If **NOT NULL** or **DEFAULT** is specified for a column of the record type created in a package, a compilation error is reported. Variables created by accessing the package.rec type do not support **NOT NULL** and **DEFAULT**.

3. If **NOT NULL** or **DEFAULT** is specified for a column of another type with a nested record type, a compilation error is reported. For a variable nested with the record type, the record element of the variable does not support **NOT NULL** and **DEFAULT**.

system_view_version

Parameter description: Determines the version of the system view. For details, see **Table 1 System view version parameters**. All versions are backward compatible. For example, when **system_view_version** is set to **3**, all features of version 2 and version 1 are also supported.

Parameter type: integer.

Unit: none

Value range: 0 to 9999

Default value: **0**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: In the upgrade scenario, do not change the value. In the installation scenario, set this parameter to the latest version listed in **Table 1 System view version parameters**.

Table 14-11 System view version parameters

Value	Description
0	Default behavior
1	The value of **PREPARECOUNT** in the V$GLOBAL_TRANSACTION view changes from **NULL** to **0** when no prepared transaction exists in GaussDB.

14.3.16.2 Platform and Client Compatibility

Many platforms use the database system. External compatibility of the database system provides great convenience for platforms.

a_format_date_timestamp

Parameter description: Specifies whether to return the date and time.

In A-compatible mode, when a transaction is started, the functions current_date, current_timestamp, and localtimestamp return the timestamp when the current SQL statement is started.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: Returns the timestamp when the current SQL statement is started.
- **off**: Returns the date or date and time when the transaction is started.

Default value: **off**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. This parameter can be enabled when the system time needs to be returned when a transaction is started.

convert_string_to_digit

Parameter description: Specifies the implicit conversion priority, which determines whether to preferentially convert strings into numbers.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that strings are preferentially converted into numbers.
- **off** indicates that strings are not preferentially converted into numbers.

Default value: **on**

NOTICE

Adjusting this parameter will change the internal data type casting rule and cause unexpected behaviors. Exercise caution when performing this operation.

nls_timestamp_format

Parameter description: Specifies the default timestamp format.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string.

Default value: **DD-Mon-YYYY HH:MI:SS.FF AM**

nls_timestamp_tz_format

Parameter description: Specifies the default timestamp with time zone format.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. The supported formats are the same as those of **nls_timestamp_format**.

Default value: **DD-Mon-YYYY HH:MI:SS.FF AM**

NOTE

This parameter is valid only when **a_format_version** is set to **10c** and **a_format_dev_version** is set to **s1**.

nls_nchar_characterset

Parameter description: Sets the national character set, which is used together with the nchr(cvalue int|bigint) system function. The value of this parameter is an enumerated string.

Parameter type: string.

Unit: none

Value range: **'AL16UTF16'** and **'UTF8'** (case insensitive)

Default value: **'AL16UTF16'**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

NOTE

- The database does not support the national character set. This parameter is used only to be compatible with the A database and obtain the national character set for users.
- This GUC parameter applies only to the nchr(cvalue int|bigint) function.

group_concat_max_len

Parameter description: This parameter is used together with the GROUP_CONCAT function to limit the length of the return value. If the length exceeds the limit, the exceeded part of the return value is truncated.

Parameter type: integer.

Unit: none

Value range: 0 to 9223372036854775807. Currently, the maximum length that takes effect is 1073741823. If the length exceeds this limit, the out of memory error is reported.

Default value: 1024

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

max_function_args

Parameter description: Specifies the maximum number of parameters allowed for a function.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

Value range: an integer.

Default value: **8192**

max_subpro_nested_layers

Parameter description: Specifies the maximum nesting depth of nested subprograms.

Parameter type: integer.

Unit: none

Value range: 0 to 100

NOTE

When this parameter is set to **0**, nested subprograms are not allowed.

Default value: **3**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Set this parameter based on the maximum nesting depth.

transform_null_equals

Parameter description: Specifies whether expressions of the form expr = NULL (or NULL = expr) are treated as expr IS NULL. They return true if expr evaluates to the **NULL** value, and false otherwise.

- The correct SQL-standard-compliant behavior of **expr = NULL** is to always return **NULL** (unknown).
- Filtered forms in Microsoft Access generate queries that appear to use **expr = NULL** to test for null values. If you enable this option, you can use this API to access the database.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that expressions of the form **expr = NULL** (or **NULL = expr**) are treated as expr **IS NULL**.
- **off** indicates that **expr = NULL** always returns **NULL** (unknown).

Default value: **off**

NOTE

New users are always confused about the semantics of expressions involving **NULL** values. Therefore, **off** is used as the default value.

support_extended_features

Parameter description: Specifies whether extended database features are supported.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that extended database features are supported.
- **off** indicates that extended database features are not supported.

Default value: **off**

enable_extension

Parameter description: Specifies whether to support the creation of database extensions.

Parameter type: Boolean.

Unit: none

Value range:

- **on** indicates that the creation of database extensions is supported.
- **off** indicates that the creation of database extensions is not supported.

Default value: **off**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

NOTICE

The extended function is for internal use only. You are advised not to use it.

sql_compatibility

Parameter description: Specifies the type of mainstream database with which the SQL syntax and statement behavior of the database is compatible. This is an INTERNAL parameter. It can be viewed but cannot be modified.

Value range: enumerated values

- **A** indicates that the database is compatible with the Oracle database.
- **B** indicates that the database is compatible with the MySQL database.
- **C** indicates that the database is compatible with the Teradata database.
- **PG** indicates that the database is compatible with the PostgreSQL database.

Default value: **A**

NOTICE

- This parameter can be set only when you run the **CREATE DATABASE** command to create a database. For details, see "SQL Reference > SQL Syntax > CREATE DATABASE" in *Developer Guide*.
- In the database, this parameter must be set to a specific value. It can be set to **A** or **B** and cannot be changed. Otherwise, the setting is not consistent with the database behavior.

b_format_behavior_compat_options

Parameter description: Specifies a B-compatible database configuration item.

Parameter type: string.

Unit: none

Value range: Currently, only B-compatible configuration items listed in **Table 1** are supported. Use commas (,) to separate multiple compatibility configuration items, for example, **set b_format_behavior_compat_options="enable_set_variables,set_session_transaction"**.

Default value: ""

NOTE

If **b_format_version** is not set to '', **b_format_behavior_compat_options** will be set to **all** and cannot be modified.

Table 14-12 B-compatible configuration items

Configuration Item	Behavior
enable_set_variables	Specifies whether to enable the enhancement of the SET syntax. • If this parameter is not set, user-defined variables and the SET [GLOBAL | SESSION] syntax are not supported. • If this parameter is set, the preceding syntax is supported in the B-compatible mode, for example, *set @v1 = 1;*.
set_session_transaction	Specifies whether to enable the SET SESSION TRANSACTION syntax. • If this parameter is not set, SET SESSION TRANSACTION is equivalent to SET LOCAL TRANSACTION. • If this parameter is set, the preceding syntax can be used in the B-compatible mode to modify the transaction features in the current session.

Configuration Item	Behavior
enable_modify_column	Specifies whether to enable the ALTER TABLE MODIFY syntax. • If this parameter is not set, running **ALTER TABLE table_name MODIFY column_name data_type;** can change only the data type of the column. • If this parameter is set, running **ALTER TABLE table_name MODIFY column_name data_type;** can change the entire column definition.
default_collation	Specifies whether to enable forward compatibility of the default collation. • If this item is not specified, the character set or collation of a column of the character type is not explicitly specified, and the collation of a table is empty, the column uses the default collation. • If this item is specified, the collation of a column of the character type inherits the collation of a table (if it is not empty). If the collation of a table is empty, the default collation corresponding to the database is used.
all	Determines whether to enable all syntax. **all** cannot be specified together with other configuration items. In the table, specifying all configuration items except **all** that are separated by commas is equivalent to specifying **all**.

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Determines whether some B-compatible features are available. If you want to modify this parameter, make sure you understand its meaning and modify it with caution to avoid risks caused by misoperations.

m_format_behavior_compat_options

Parameter description: Specifies the configuration items of an M-compatible database.

Parameter type: string.

Unit: none

Value range: Currently, only **Table 14-13** are supported. Compatibility configuration items are separated by commas (,).

Default value: **""**

Table 14-13 M-compatible configuration items

Configuration Item	Behavior
enable_escape_string	Specifies whether escape character control is enabled. • If this item is not set, escape character behaviors are controlled by the GUC parameters **standard_conforming_strings**, **escape_string_warning**, and **backslash_quote** related to the GaussDB escape character. • If this item is set, all MySQL escape characters except '\0' are supported by default. In addition, the output behavior of '\b', '\r', and '\Z' on the gsql client changes to be the same as that on the MySQL client.

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-2**.

Setting suggestion: none.

behavior_compat_options

Parameter description: Specifies the database compatibility configuration item.

Parameter type: string.

Unit: none

Value range: Currently, only compatibility configuration items listed in **Table 14-14** are supported. Use commas (,) to separate multiple compatibility configuration items, for example, **set behavior_compat_options='end_month_calculate,display_leading_zero'**.

Default value: **"forbid_package_function_with_prefix, enable_bpcharlikebpchar_compare,enable_crosstype_integer_operator"**

Table 14-14 Compatibility configuration items

Configuration Item	Behavior
display_leading_zero	Specifies how floating-point numbers are displayed. It determines the display of zeros before the decimal point of all character string types (such as char, character, nchar, varchar, character varying, varchar2, nvarchar2, text, and clob) and any precision types (such as float4, float8, and numeric) in the numeric type. In addition, the length of the calculated number is displayed synchronously. ● If this item is not specified, for a non-zero decimal number between –1 and +1, the 0 before the decimal point is not displayed. For example: `gaussdb=# select 0.1231243 as a, 0.1231243::numeric as b,0.1231243::integer(10,3) as c, length(0.1242343) as d;` ` a	b	c	d` `----------+----------+------+---` `.1231243	.1231243	.123	8` `(1 row)` ● If this item is specified, for a non-zero decimal number between –1 and +1, the 0 before the decimal point is displayed. For example: `gaussdb=# select 0.1231243 as a, 0.1231243::numeric as b,0.1231243::integer(10,3) as c, length(0.1242343) as d;` ` a	b	c	d` `----------+----------+-------+---` `0.1231243	0.1231243	0.123	9` `(1 row)`
end_month_calculate	Specifies the calculation logic of the add_months function. Assume that the two parameters of the add_months function are **param1** and **param2**, and that the month of **param1** and **param2** is **result**. ● If this item is not specified, and the **Day** of **param1** indicates the last day of a month shorter than **result**, the **Day** in the calculation result will equal that in **param1**. For example: `gaussdb=# select add_months('2018-02-28',3) from sys_dummy;` `add_months` `---------------------` `2018-05-28 00:00:00` `(1 row)` ● If this item is specified, and the **Day** of **param1** indicates the last day of a month shorter than **result**, the **Day** in the calculation result will equal that in **result**. For example: `gaussdb=# select add_months('2018-02-28',3) from sys_dummy;` `add_months` `---------------------` `2018-05-31 00:00:00` `(1 row)`

Configuration Item	Behavior
compat_analyze_sample	Specifies the sampling behavior of the **ANALYZE** operation. If this item is specified, the sample collected by the ANALYZE operation will be limited to around 30,000 records, controlling database node memory consumption and maintaining the stability of ANALYZE.
bind_schema_tablespace	Binds a schema with the tablespace with the same name. If a tablespace name is the same as *sche_name*, **default_tablespace** will also be set to *sche_name* if **search_path** is set to *sche_name*.
bind_procedure_searchpath	Specifies the search path of the database objects in a stored procedure for which no schema name is specified. If no schema name is specified for a stored procedure, the schema to which the stored procedure belongs is searched preferentially. If the stored procedure is not found, the following operations are performed: ● If this item is not specified, the system reports an error and exits. ● If this item is specified, the search continues based on the settings of **search_path**. If the issue persists, the system reports an error and exits.
correct_to_number	Specifies the compatibility of the to_number() result. ● If this item is not specified, the result of the to_number() function is the same as that in the A database by default. `gaussdb=# select '' AS to_number_14, to_number('34,50','999,99');` `ERROR:  invalid data.` `CONTEXT:  referenced column: to_number` ● If this item is specified, the result of the to_number() function is the same as that of pg11. `gaussdb=# select '' AS to_number_14, to_number('34,50','999,99');` ` to_number_14 | to_number` `--------------+-----------` `              |      3450` `(1 row)`

Configuration Item	Behavior
unbind_divide_bound	Specifies the range check on the result of integer division. ● If this item is not specified, the range of the division result is verified. For example, an out-of-bounds error is reported because the output result of *INT_MIN*/**(–1)** is greater than *INT_MAX*. `gaussdb=# select (-2147483648)::int4 / (-1)::int4;` `ERROR:  integer out of range` ● If this item is specified, the range of the division result does not need to be verified. For example, the output result of *INT_MIN*/**(–1)** is *INT_MAX*+**1**. `gaussdb=# select (-2147483648)::int4 / (-1)::int4;` `  ?column?` `------------` ` 2147483648` `(1 row)`
convert_string_digit_to_numeric	Determines whether to convert numeric constants of the character string type to those of the numeric type before these two types are compared. ● If this item is not specified, the numeric constants of the character string type are not converted to those of the numeric type. ● If this item is specified, the numeric constants of the character string type are converted to those of the numeric type. `gaussdb=# create table test1 (c1 int, c2 varchar);` `gaussdb=# insert into test1 values (2, '1.1');` `gaussdb=# set behavior_compat_options='';` `gaussdb=# select * from test1 where c2 > 1;` `ERROR:  invalid input syntax for type bigint: "1.1"` `gaussdb=# set behavior_compat_options='convert_string_digit_to_numeric';` `gaussdb=# select * from test1 where c2 > 1;` ` c1 | c2` `----+-----` `  2 | 1.1` `(1 row)`
return_null_string	Specifies how to display the empty result (empty string '') of the lpad() and rpad() functions. ● If this item is not specified, the empty string is displayed as **NULL**. `gaussdb=# select length(lpad('123',0,'*')) from sys_dummy;` `length` `--------` `(1 row)` ● If this item is specified, the empty string is displayed as single quotation marks (''). `gaussdb=# select length(lpad('123',0,'*')) from sys_dummy;` `length` `--------` `0` `(1 row)`

Configuration Item	Behavior
compat_concat_variadic	Specifies the compatibility of variadic results of the concat() and concat_ws() functions. The B database does not have the variadic type. Therefore, this option has no impact on the B database. ● If this item is not specified and the concat function parameter is of the variadic type, the results of the A and C databases in compatibility mode are the same by default. `gaussdb=# select concat(variadic NULL::int[]) is NULL;` ` ?column?` `----------` ` t` `(1 row)` ● If this item is specified and the concat function parameter is of the variadic type, different result formats of the A and C databases in compatibility mode are retained. `--In the A database:` `gaussdb=# select concat(variadic NULL::int[]) is NULL;` ` ?column?` `----------` ` t` `(1 row)` `--In the C database:` `gaussdb=# select concat(variadic NULL::int[]) is NULL;` ` ?column?` `----------` ` f` `(1 row)`
merge_update_multi	When MERGE INTO ... WHEN MATCHED THEN UPDATE (see "SQL Reference > SQL Syntax > MERGE INTO" in *Developer Guide*) and INSERT ... ON DUPLICATE KEY UPDATE (see "SQL Reference > SQL Syntax > INSERT" in *Developer Guide*) are used, it controls the UPDATE behavior if a piece of target data in the target table conflicts with multiple pieces of source data. ● If this item is specified and the preceding scenario exists, the system performs multiple UPDATE operations on the conflicting row. ● If this item is not specified (the default value is retained), an error is reported, indicating that the MERGE or INSERT operation fails.

Configuration Item	Behavior
plstmt_implicit_savepoint	Determines whether the execution of an UPDATE statement in a stored procedure has an independent subtransaction. If this parameter is set, the implicit savepoint is enabled before executing each UPDATE statement in the stored procedure, and the subtransaction is rolled backed to the latest savepoint in the EXCEPTION block by default, ensuring that only the modification of failed statements is rolled back. This option is used to be compatible with the EXCEPTION behavior of the ORA database.
hide_tailing_zero	Configuration item for numeric display. ● If this item is specified, the trailing zeros after the decimal point are hidden in all scenarios where numeric values are output, even if the precision format is specified. ● If this item is not specified, numeric data is displayed in the specified precision. `gaussdb=# set behavior_compat_options='hide_tailing_zero';` `gaussdb=# select cast(123.123 as numeric(15,10)) as a, to_char(cast(123.123 as numeric(15,10)), '999D999999');` ` a	to_char` `---------+----------` ` 123.123	123.123` `(1 row)` `gaussdb=# set behavior_compat_options='';` `gaussdb=# select cast(123.123 as numeric(15,10)) as a, to_char(cast(123.123 as numeric(15,10)), '999D999999');` ` a	to_char` `----------------+-------------` ` 123.1230000000	123.123000` `(1 row)`

Configuration Item	Behavior
rownum_type_compat	Specifies the ROWNUM type. The default value is **BIGINT**. After this parameter is specified, the value is changed to **NUMERIC**. gaussdb=# set behavior_compat_options=''; gaussdb=# create table tb_test(c1 int,c2 varchar2,c3 varchar2); gaussdb=# insert into tb_test values(1,'a','b'); gaussdb=# create or replace view v_test as select rownum from tb_test; gaussdb=# \d+ v_test View "public.v_test" Column | Type | Modifiers | Storage | Description --------+--------+-----------+---------+------------- rownum | bigint | | plain | View definition: SELECT ROWNUM AS "rownum" FROM tb_test; gaussdb=# set behavior_compat_options = 'rownum_type_compat'; gaussdb=# create or replace view v_test1 as select rownum from tb_test; gaussdb=# \d+ v_test1 View "public.v_test1" Column | Type | Modifiers | Storage | Description --------+---------+-----------+---------+------------- rownum | numeric | | main | View definition: SELECT ROWNUM AS "rownum" FROM tb_test;
aformat_null_test	Specifies the logic for checking whether **rowtype** is not null. When this parameter is set, if one column in a row is not empty, **true** is returned for checking whether **rowtype** is not null. When this parameter is not set, if all columns in a row are not empty, true is returned for checking whether **rowtype** is not null. This parameter has no influence on checking whether **rowtype** is not null. gaussdb=# set behavior_compat_options='aformat_null_test'; gaussdb=# select r, r is null as isnull, r is not null as isnotnull from (values (1,row(1,2)), (1,row(null,null)), (1,null), (null,row(1,2)), (null,row(null,null)), (null,null)) r(a,b); r | isnull | isnotnull -------------+--------+----------- (1,"(1,2)") | f | t (1,"(,)") | f | t (1,) | f | t (,"(1,2)") | f | t (,"(,)") | f | t (,) | t | f (6 rows) gaussdb=# set behavior_compat_options=''; gaussdb=# select r, r is null as isnull, r is not null as isnotnull from (values (1,row(1,2)), (1,row(null,null)), (1,null), (null,row(1,2)), (null,row(null,null)), (null,null)) r(a,b); r | isnull | isnotnull -------------+--------+----------- (1,"(1,2)") | f | t (1,"(,)") | f | t (1,) | f | f (,"(1,2)") | f | f (,"(,)") | f | f (,) | t | f (6 rows)

Configuration Item	Behavior
aformat_regexp_match	Determines the matching behavior of regular expression functions. When this parameter is set and **sql_compatibility** is set to **A** or **B** or in an M-compatible database, the options supported by the **flags** parameter of the regular expression are changed as follows: 1. By default, the character '\n' cannot be matched. 2. If **flags** contains the **n** option, the period (.) can match the character '\n'. 3. The regexp_replace(source, pattern replacement) function replaces all matching substrings. 4. regexp_replace(source, pattern, replacement, flags) returns **null** when the value of **flags** is **''** or **null**. Otherwise, the meanings of the options supported by the **flags** parameter of the regular expression are as follows: 1. By default, the character '\n' can be matched. 2. The **n** option in **flags** indicates that the multi-line matching mode is used. 3. The regexp_replace(source, pattern replacement) function replaces only the first matched substring. 4. If the value of **flags** is **''** or **null**, the return value of regexp_replace(source, pattern, replacement, flags) is the character string after replacement.
compat_cursor	Determines the compatibility behavior of implicit cursor states. If this parameter is set and the A-compatible mode is used, the effective scope of implicit cursor states (SQL %FOUND, SQL%NOTFOUND, SQL%ISOPNE, and SQL %ROWCOUNT) is extended from only the currently executed function to all subfunctions called by this function.
proc_outparam_override	Determines the overloading of output parameters of a stored procedure. After this parameter is enabled, the stored procedure can be properly created and called even if only the output parameters of the stored procedure are different. Currently, this parameter can be used only when gsql and JDBC are used to connect to the database. If this parameter is enabled for other tools to connect to the database, stored procedures with the **out** parameter cannot be called. It supports the functions that contain the **out** output parameter and returns data of the record type. Besides, a value is assigned to the **out** parameter.

Configuration Item	Behavior
proc_implicit_for_loop_variable	Determines the behavior of the FOR_LOOP query statement in a stored procedure. When this parameter is set, if *rec* has been defined in the FOR rec IN query LOOP statement, the defined *rec* variable is not reused but a new variable is created. Otherwise, the defined *rec* variable is reused and no variable is created.
allow_procedure_compile_check	Controls the compilation check of the SELECT and OPEN CURSOR statements in a stored procedure. If this parameter is set, when the SELECT, OPEN CURSOR FOR, CURSOR%rowtype, or FOR rec IN statement is executed in a stored procedure, the stored procedure cannot be created if the queried table does not exist, and the compilation check of the trigger function is not supported. If the queried table exists, the stored procedure is successfully created. Note: When creating an encrypted function, you need to disable **allow_procedure_compile_check**.

Configuration Item	Behavior
char_coerce_compat	Controls the behavior when char(n) types are converted to other variable-length string types. This parameter is valid only when the **sql_compatibility** parameter is set to **A**. After this parameter is enabled, spaces at the end are not omitted in implicit conversion, explicit conversion, or conversion by calling the text(bpchar) function. ● If this item is specified, spaces at the end are not omitted in conversion. In addition, if the length of the char(n) type exceeds that of other variable-length string types, an error is reported. ● If this item is not specified, spaces at the end are omitted when the char(n) type is converted to another variable-length string type. gaussdb=# set behavior_compat_options=''; gaussdb=# create table tab_1(col1 varchar(3)); gaussdb=# create table tab_2(col2 char(3)); gaussdb=# insert into tab_2 values(' '); gaussdb=# insert into tab_1 select col2 from tab_2; gaussdb=# select * from tab_1 where col1 is null; col1 ------ (1 row) gaussdb=# select * from tab_1 where col1=' '; col1 ------ (0 rows) gaussdb=# delete from tab_1; gaussdb=# set behavior_compat_options = 'char_coerce_compat'; gaussdb=# insert into tab_1 select col2 from tab_2; gaussdb=# select * from tab_1 where col1 is null; col1 ------ (0 rows) gaussdb=# select * from tab_1 where col1=' '; col1 ------ (1 row)

Configuration Item	Behavior
truncate_numeric_tail_zero	Configuration item for numeric display. ● If this item is specified, the trailing zeros after the decimal point are hidden in all numeric output scenarios except to_char(numeric, format). ● If this item is not specified, numeric data is displayed in the default precision. For example: gaussdb=# set behavior_compat_options='truncate_numeric_tail_zero'; gaussdb=# select cast(123.123 as numeric(15,10)) as a, to_char(cast(123.123 as numeric(15,10)), '999D999999'); a | to_char ---------+------------- 123.123 | 123.123000 (1 row) gaussdb=# set behavior_compat_options=''; gaussdb=# select cast(123.123 as numeric(15,10)) as a, to_char(cast(123.123 as numeric(15,10)), '999D999999'); a | to_char ----------------+------------- 123.1230000000 | 123.123000 (1 row)
plsql_security_definer	After this parameter is enabled, the definer permission is used by default when a stored procedure is created.

Configuration Item	Behavior
plpgsql_dependency	If this parameter is set, a function, stored procedure, or package containing undefined objects can be created. You can query the dependency in GS_DEPENDENCIES and GS_DEPENDENCIES_OBJ. If this parameter is enabled, when creating a PL/SQL object, the OID that depends on the PL/SQL object is automatically updated. Dependency can be established in the following scenarios: 1. A function appears at the position of the dependency type and parameter default value in a function header. 2. Package, type in the function, and variable dependency type. 3. Variable declaration and variable value assignment depend on variables in other packages. 4. In the function body, function A is called in the right value expression of the function call or assignment statement. If the input and output parameters of function A contain function B, no dependency is established for function B. For example, **functionA(functionB())**. Only the dependency for function A is established. 5. Dependency function in a view. Dependency cannot be established in the following scenarios: 1. A type in a schema depends on other types. 2. Dependency on functions, variables, tables, and views in SQL statements. For example, dependency is not recorded for **select id into var1 from table1 join view1 on table1.id = pkg1.var1; table1,view1,pkg1**. 3. Dependency on functions, variables, tables, and views in a view. Notes: 1. When PL/SQL objects are concurrently created, deadlocks may occur if contention occurs among the objects. 2. If the objects to be modified exist in **gs_dependencies** and **gs_dependencies_obj**, you cannot rename the objects. 3. When a function, stored procedure, or package depends on a synonym, the synonym must be created in advance. OIDs cannot be maintained manually. 4. This parameter is used together with the **plsql_expression_check** parameter of **plsql_compile_check_options** to provide complete

Configuration Item	Behavior
	verification and dependency creation functions. For details, see the description of the **plsql_expression_check** parameter. 5. Dependency types, tables, and functions do not exist at the beginning. They exist later through renaming. OIDs cannot be proactively maintained. 6. The package function is overloaded, the number of input and output parameters is the same, but the types are different. In addition, the input and output parameters depend on external types, and the sequence numbers of external types in the input and output parameters are the same. In this case, an error may be reported when a type is deleted. 7. The input and output parameters of the function and stored procedure depend on the external type **a.b**. The external type must be created in advance. After deleting a type, you need to rebuild the corresponding function and stored procedure. Example: **proc1(var1 a.b, var2 a.b %type, var3 a.b %rowtype)**. 8. For a specific dependency, for example, when you create function, stored procedure, or view A, the default values of the input and output parameters of A depend on the defined function B: ● When the **plpgsql_dependency** parameter is enabled, even if the pg_depend table records the dependency data of A and B, A is not deleted when B is deleted. ● When the **plpgsql_dependency** parameter is disabled, A is deleted when B is deleted because the pg_depend table records the dependency data of A and B. ● If the gs_dependencies and gs_dependencies_obj tables contain dependent data, the dependent data will be deleted regardless of whether **plpgsql_dependency** is enabled or not. The dependencies are as follows: ● The default values of input and output parameters of the function or stored procedure A depend on function B. ● The input and output parameters or return value of the function or stored procedure A depends on type B. ● The public variables of package A depend on the public variables of package B. ● View A depends on function B.

Configuration Item	Behavior
	9. Leave this parameter unchanged throughout the operation. If this parameter is disabled, OIDs that depend on the PL/SQL object are not proactively maintained, and data in gs_dependencies and gs_dependencies_obj is not updated. As a result, residual data exists in the gs_dependencies and gs_dependencies_obj tables. 10. If this parameter is enabled or disabled during the operation, data in the gs_dependencies catalog will be inconsistent. Therefore, you are advised not to do so. If this parameter is disabled, the following alarm information is displayed: WARNING: Disable parameter plpgsql_dependency. Check whether the data in the system table is normal. Please see Centralized-Administrator Guide->Configuring Running Parameters->GUC Parameters->Version and Platform Compatibility->Platform and Client Compatibility->plpgsql_dependency. DETAIL: Disable parameter plpgsql_dependency. see the description of the plpgsql_dependency parameter For example: Enable the parameter. set behavior_compat_options='plpgsql_dependency'; create type type1 as (c1 int, c2 int); CREATE OR REPLACE function fun1(var int) return int is var type1; begin return 1; end; / The data in the system catalog is as follows: gaussdb=# select * from gs_dependencies; schemaname | packagename | refobjpos | refobjoid | objectname ------------+-------------+-----------+----------- +---------------------- public | null | 8 | 115940 | fun1(pg_catalog.int4) (1 row) gaussdb=# select oid,* from gs_dependencies_obj order by oid;

Configuration Item	Behavior
	oid | schemaname | packagename | type | name | objnode --------+------------+-------------+------+------- +---------- ---------- ---------- 115940 | public | null | 3 | type1 | {DependenciesType :typType c :typCategory C :isRel false :attrInfo c1:pg_catalog.int4,c2:pg_catalog.int4, :elemTypName <> :idxByTypName <>} (1 row) Disable the parameter. set behavior_compat_options=''; create type type2 as (c1 int, c2 int); CREATE OR REPLACE function fun2(var int) return int is var type2; begin return 1; end; / The data in the system catalog is as follows: gaussdb=# select * from gs_dependencies; schemaname | packagename | refobjpos | refobjoid | objectname ------------+-------------+-----------+-----------+------------ (0 rows) gaussdb=# select oid,* from gs_dependencies_obj order by oid; oid | schemaname | packagename | type | name | objnode -----+------------+-------------+------+------+--------- (0 rows) If the parameter is disabled during the operation, you are advised to enable the parameter to rebuild the updated functions, stored procedures, and packages. Expected data is displayed in gs_dependencies. Rebuild **fun2**. Enable the parameter. set behavior_compat_options='plpgsql_dependency';

Configuration Item	Behavior
	CREATE OR REPLACE function fun2(var int) return int is var type2; begin return 1; end; / gaussdb=# select * from gs_dependencies; schemaname | packagename | refobjpos | refobjoid | objectname ------------+-------------+-----------+----------- +---------------------- public | null | 8 | 140563 | fun2(pg_catalog.int4) (1 row) gaussdb=# select oid,* from gs_dependencies_obj order by oid; oid | schemaname | packagename | type | name | objnode --------+------------+-------------+------+------- +-- -- ---------- 140563 | public | null | 3 | type2 | {DependenciesType :typType c :typCategory C :isRel false :attrInfo c1:pg_catalog.int4,c2:pg_catalog.int4, :elemTypName <> :idxByTypName <>} (1 row)
disable_rewrite_nesttable	If this parameter is enabled, rewriting columns of the tableof type in the pg_type table will be disabled. That is, when you read the pg_type table, the actual stored value of the tableof type is displayed.
skip_insert_gs_source	If this parameter is enabled, no data is inserted into the dbe_pldeveloper.gs_source table when the PL/SQL objects are created.

Configuration Item	Behavior
disable_emptystr2null	In the A-compatible parameter binding scenario, if this parameter is enabled, the function of converting the values of parameters of the character type from an empty string to **null** by default, if any, is disabled. text, clob, blob, raw, bytea, varchar, nvarchar2, bpchar, char, name, byteawithoutorderwithqualcol, and byteawithoutordercol. This parameter is reserved for emergency. Do not set it unless necessary.
select_into_return_null	This parameter takes effect only in PG-compatible mode. If it is enabled, **NULL** values can be assigned to the variables in the stored procedure statement **SELECT** *select_expressions* **INTO [STRICT]** *target* **FROM...** without specifying **STRICT** when the query result is empty.

Configuration Item	Behavior
proc_uncheck_default_param	When a function is called, the system does not check whether the default parameter is omitted. ● If this item is not specified and a function with default parameters is called, input parameters are added to the function from left to right. If inputs of non-default parameters are missing, an error is reported. For example: gaussdb=# create or replace function test(f1 int, f2 int default 20, f3 int, f4 int default 40, f5 int default 50) return int gaussdb-# as gaussdb$# begin gaussdb$# raise info 'f1:%',f1; gaussdb$# raise info 'f2:%',f2; gaussdb$# raise info 'f3:%',f3; gaussdb$# raise info 'f4:%',f4; gaussdb$# raise info 'f5:%',f5; gaussdb$# return 1; gaussdb$# end; gaussdb$# / CREATE FUNCTION gaussdb=# select test(1,2); ERROR: function test(integer, integer) does not exist LINE 1: select test(1,2); ^ HINT: No function matches the given name and argument types. You might need to add explicit type casts. CONTEXT: referenced column: test ● If this item is specified and a function with default parameters is called, input parameters are added to the function from left to right. The number of defaulted inputs depends on the number of default parameters. If an input of a non-default parameter is missing, the previous default value is used to fill this parameter. For example: gaussdb=# create or replace function test(f1 int, f2 int default 20, f3 int, f4 int default 40, f5 int default 50) return int gaussdb-# as gaussdb$# begin gaussdb$# raise info 'f1:%',f1; gaussdb$# raise info 'f2:%',f2; gaussdb$# raise info 'f3:%',f3; gaussdb$# raise info 'f4:%',f4; gaussdb$# raise info 'f5:%',f5; gaussdb$# return 1; gaussdb$# end; gaussdb$# / CREATE FUNCTION gaussdb=# select test(1,2); INFO: f1:1 CONTEXT: referenced column: test INFO: f2:2 CONTEXT: referenced column: test INFO: f3:20 CONTEXT: referenced column: test INFO: f4:40 CONTEXT: referenced column: test INFO: f5:50 CONTEXT: referenced column: test test

Configuration Item	Behavior
	------ 1 (1 row) As shown above, f3 is filled with an incorrect default value. **WARNING** In this scenario, a non-default parameter is filled with the previous default value.
dynamic_sql_compat	After this parameter is enabled: 1. Duplicate parameters in the SQL statement template are not regarded as the same parameter, but match variables in the USING clause in sequence. 2. If a stored procedure is called during dynamic statement execution, the IN and OUT attributes in a stored procedure and the USING clause are not checked. **CAUTION** If a stored procedure is called when a dynamic statement executes an anonymous block statement, only the **IN** parameters are corrected. If **OUT** parameters need to be checked, set the **proc_outparam_override** parameter.
dynamic_sql_check	After this parameter is enabled, an error is reported during dynamic statement execution if the number of different template parameters in the dynamic statement template SQL is different from that of variables in the USING clause. **CAUTION** ● If the **dynamic_sql_compat** option is enabled, the **dynamic_sql_check** option does not take effect. ● If a stored procedure is called when a dynamic statement executes an anonymous block statement, only the **IN** parameters are checked. If **OUT** parameters need to be checked, set the **proc_outparam_override** parameter. ● If a stored procedure is called when a dynamic statement executes an anonymous block statement and the parameter is enabled, the **IN** and **OUT** attributes in a stored procedure and the USING clause are not checked.

Configuration Item	Behavior
enable_funcname_with_argsname	If the parameter is enabled, the projection alias displays the complete function when SELECT is used to call a function. • If this item is not specified, the projection alias displays only the function name when SELECT is used to call a function. For example: `gaussdb=# select power(2,3);` ` power` `-------` `     8` `(1 row)` `gaussdb=# select count(*) from db_ind_columns;` ` count` `-------` `   611` `(1 row)` `gaussdb=# select count(index_name) from db_ind_columns;` ` count` `-------` `   611` `(1 row)` `gaussdb=# SELECT left('abcde', 2);` ` left` `------` ` ab` `(1 row)` `gaussdb=# SELECT pg_client_encoding();` ` pg_client_encoding` `--------------------` ` UTF8` `(1 row)` • If this item is specified, the projection alias displays the complete function when SELECT is used to call a function. For example: `gaussdb=# set behavior_compat_options =` `'enable_funcname_with_argsname';` `SET` `gaussdb=# select power(2,3);` ` power(2,3)` `------------` `          8` `(1 row)` `gaussdb=# select count(*) from db_ind_columns;` ` count(*)` `----------` `      611` `(1 row)` `gaussdb=# select count(index_name) from db_ind_columns;` ` count(index_name)` `-------------------` `              611` `(1 row)` `gaussdb=# SELECT left('abcde', 2);` ` left('abcde',2)` `-----------------`

Configuration Item	Behavior
	ab (1 row) gaussdb=# SELECT pg_client_encoding(); pg_client_encoding() ---------- UTF8 (1 row) **CAUTION** ● Currently, only func_name(args_list), func_name(), and func_name(*) can be used to display complete functions and the arguments can only be character strings, numeric values, column names, and functions. A function name can contain a schema or package name. The parameter cannot contain other clauses (such as the ORDER BY clause) or be an expression. The parameter can contain only the DISTINCT keyword. If the parameter contains other keywords, complete functions cannot be displayed. ● Some special functions do not support displaying projection alias, including COLLATION FOR, CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP, DBTIMEZONE, LOCALTIME, LOCALTIMESTAMP, SYSDATE, SESSIONTIMEZONE, ROWNUM, CURRENT_ROLE, CURRENT_USER, SESSION_USER, USER, CURRENT_CATALOG, CURRENT_SCHEMA, CAST, EXTRACT, TIMESTAMPDIFF, OVERLAY, POSITION, SUBSTRING, TREAT, TRIM, NULLIF, NVL, NVL2, COALESCE, GREATEST, LEAST, LNNVL, REGEXP_LIKE, and XML functions. ● If some secure encryption and decryption functions and masking functions are displayed completely by projection aliases, it may bring security problems. Therefore, only function names are displayed here, including: gs_encrypt_aes128, gs_decrypt_aes128, gs_encrypt, gs_decrypt, gs_encrypt_bytea, gs_decrypt_bytea, aes_encrypt, aes_decrypt, pg_create_physical_replication_slot_extern, dblink_connect, creditcardmasking, basicemailmasking, fullemailmasking, alldigitsmasking, shufflemasking, randommasking, regexpmasking, and gs_digest. ● Parameter transfer using => is not supported for the projection alias to display the complete function. The projection alias cannot contain double quotation marks (""), for example, **select "power"(2,3)**. ● To enable the projection alias to display the complete function, this function is not affected by parameters such as removing 0 at the end.

Configuration Item	Behavior
proc_outparam_transfer_length	After this parameter is enabled, the length of the **out** output parameter can be transferred in the stored procedure and function, and an error is reported in the inner stored procedure or function. For example: gaussdb=# SET behavior_compat_options='proc_outparam_override,proc_outparam_transfer_length'; SET gaussdb=# CREATE OR REPLACE PROCEDURE out_param_test1(m in int, v inout varchar2,v1 inout varchar2) is gaussdb$# begin gaussdb$# v := 'aaaddd'; gaussdb$# v1 := 'aaaddd'; gaussdb$# end; gaussdb$# / CREATE PROCEDURE gaussdb=# CREATE OR REPLACE PROCEDURE call_out_param_test1 is gaussdb$# v varchar2(5) := 'aabbb'; gaussdb$# v1 varchar2(6) := 'aabb'; gaussdb$# begin gaussdb$# out_param_test1(5,v,v1); gaussdb$# end; gaussdb$# / CREATE PROCEDURE gaussdb=# CALL call_out_param_test1(); ERROR: value too long for type character varying(5) CONTEXT: PL/SQL function out_param_test1(integer,character varying,character varying) line 3 at assignment PL/SQL function call_out_param_test1() line 4 at SQL statement
varray_compat	After this parameter is enabled: 1. The behaviors of assigning values to elements of the array type and reading elements change. 2. The behavior of calling related array functions changes. 3. After this parameter is enabled, the maximum capacity behavior difference of the varray type is verified. For details and examples, see "Stored Procedure > Arrays, Collections, and Records > Arrays" in *Developer Guide*. **CAUTION** • Do not switch and use this parameter in the same session. • After a database is upgraded: • For array types that have been defined in stored procedures or packages, enabling the **varray_compat** parameter does not take effect. You need to rebuild the array type. That is, you need to rebuild the stored procedure or package by performing the CREATE OR REPLACE or DROP+CREATE operation. The parameter takes effect only after the array type is rebuilt. • After the **plpgsql_dependency** parameter is enabled, if the content of the CREATE OR REPLACE PACKAGE/CREATE OR REPLACE PACKAGE BODY operation is the same as that of the original package or package body, the original array type will not be rebuilt. Enabling the **varray_compat** parameter does not take effect.

Configuration Item	Behavior
tableof_elem_constraints	After this parameter is enabled: 1. For the collection type, the validity of elements will be verified. 2. If the index is of the varchar collection type, the validity of the index length is also verified. For details, see "Stored Procedure > Arrays, Collections, and Records > Collections" in *Developer Guide*. **CAUTION** • Do not switch and use this parameter in the same session. • After a database is upgraded: • For collection types that have been defined in stored procedures or packages, enabling the **tableof_elem_constraints** parameter does not take effect. You need to rebuild the collection type. That is, you need to rebuild the stored procedure or package by performing the CREATE OR REPLACE or DROP+CREATE operation. The parameter takes effect only after the collection type is rebuilt. • After the **plpgsql_dependency** parameter is enabled, if the content of the CREATE OR REPLACE PACKAGE/CREATE OR REPLACE PACKAGE BODY operation is the same as that of the original package or package body, the original collection type will not be rebuilt. Enabling the **tableof_elem_constraints** parameter does not take effect. • If a collection type is created using the syntax: CREATE TYPE type_name IS TABLE OF data_type, you need to drop the type and rebuild it for the function to take effect. The following is an example of verifying the validity of elements: If this parameter is not enabled, the elements are not verified when the following stored procedure is called: `gaussdb=# CREATE OR REPLACE procedure p1 is` `gaussdb$#  type t1 is table of varchar(5);` `gaussdb$#  v t1 := t1();` `gaussdb$# begin` `gaussdb$#  v.extend();` `gaussdb$#  v(1) := '123456';` `gaussdb$#  raise info '%', v;` `gaussdb$# end;` `gaussdb$# /` `CREATE PROCEDURE` `gaussdb=# CALL p1();` `INFO:  {123456}` ` p1` `----` `(1 row)` If this parameter is enabled, the elements are verified and an error may be reported when the following stored procedure is called: `gaussdb=# SET behavior_compat_options = 'tableof_elem_constraints';` `SET` `gaussdb=# CREATE OR REPLACE procedure p1 is`

Configuration Item	Behavior
	gaussdb$# type t1 is table of varchar(5); gaussdb$# v t1 := t1(); gaussdb$# begin gaussdb$# v.extend(); gaussdb$# v(1) := '123456'; gaussdb$# raise info '%', v; gaussdb$# end; gaussdb$# / CREATE PROCEDURE gaussdb=# CALL p1(); ERROR: value too long for type character varying(5) CONTEXT: PL/SQL function p1() line 5 at assignment The following is an example of verifying the index value: If this parameter is not enabled, no error is reported when the index value exceeds the defined length. gaussdb=# CREATE OR REPLACE procedure p1 is gaussdb$# type t1 is table of int index by varchar(5); gaussdb$# v t1; gaussdb$# begin gaussdb$# v('123456') := 1; gaussdb$# raise info '%', v; gaussdb$# end; gaussdb$# / CREATE PROCEDURE gaussdb=# CALL p1(); INFO: {indexbyType:varchar,"123456"=>1} p1 ---- (1 row) If this parameter is enabled, an error is reported when the index value exceeds the defined length. gaussdb=# SET behavior_compat_options = 'tableof_elem_constraints'; SET gaussdb=# CREATE OR REPLACE procedure p1 is gaussdb$# type t1 is table of int index by varchar(5); gaussdb$# v t1; gaussdb$# begin gaussdb$# v('123456') := 1; gaussdb$# raise info '%', v; gaussdb$# end; gaussdb$# / CREATE PROCEDURE gaussdb=# CALL p1(); ERROR: value too long for type character varying(5) CONTEXT: PL/SQL function p1() line 4 at assignment

Configuration Item	Behavior
allow_rownum_alias	After this parameter is enabled, ROWNUM can be used as a column alias in SQL statements using the AS syntax. ROWNUM is used as a common identifier and cannot be used as a pseudocolumn. **WARNING** You are advised not to change the status of this parameter during service execution. When the parameter is enabled, database objects (such as table names, column names, and database names) are created using ROWNUM as the name in the database. If the parameter is disabled, ambiguity occurs and the behavior is unpredictable. When the parameter is disabled, the behavior of using ROWNUM as a pseudocolumn in the database becomes invalid after the parameter is enabled and the behavior is unpredictable.
current_sysdate	If this parameter is enabled, the current OS time is obtained when the **sysdate** command is executed. gaussdb=# set behavior_compat_options='current_sysdate'; SET gaussdb=# select sysdate; current_sysdate --------------------- 2023-06-20 20:15:27 (1 row)

<table>
<tr><th>Configuration Item</th><th>Behavior</th></tr>
<tr><td>allow_function_procedure_replace</td><td>In the default scenario, for A and PG compatibility modes, when there is a stored procedure (or function) outside the package, you cannot create a function (or stored procedure) with the same name because they are of different object types, and an error is reported. After this parameter is enabled, you can use the CREATE OR REPLACE method to replace stored procedures and functions with the same names outside the package.
<pre>
gaussdb=# create or replace function proc_test return varchar2 as
gaussdb$# begin
gaussdb$# return '1';
gaussdb$# end;
gaussdb$# /
CREATE FUNCTION
gaussdb=# create or replace procedure proc_test as
gaussdb$# begin
gaussdb$# null;
gaussdb$# end;
gaussdb$# /
ERROR: cannot change routine kind
DETAIL: "proc_test" is a function.
gaussdb=# \df+ proc_test
                                                                    List of functions
 Schema | Name | Result data type | Argument data types | Type |
Volatility | Owner | Language | Source code | Description | fencedmode |
propackage | prokind
--------+-----------+-------------------+---------------------+--------+------------
+-----------+----------+-------------+-------------+------------+------------+---------
 public | proc_test | character varying | | normal | volatile |
wangxinyu | plpgsql | DECLARE   +| | f | f | f
        | | | | | |
begin   +| | | |
        | | | | | |
return '1';+| | | |
        | | | | | |
end | | | |
(1 row)

-- Types can be replaced after the parameter is set.
gaussdb=# set behavior_compat_options='allow_function_procedure_replace';
SET
gaussdb=# create or replace procedure proc_test as
gaussdb$# begin
gaussdb$# null;
gaussdb$# end;
gaussdb$# /
CREATE PROCEDURE
gaussdb=# \df+ proc_test
                                                                   List of functions
 Schema | Name | Result data type | Argument data types | Type |
Volatility | Owner | Language | Source code | Description | fencedmode |
propackage | prokind
--------+-----------+------------------+---------------------+--------+------------
+-----------+----------+-------------+-------------+------------+------------+---------
 public | proc_test | void | | normal | volatile |
wangxinyu | plpgsql | DECLARE   +| | f | f | p
        | | | | | |
begin   +| | | |
        | | | | | |
null;   +| | | |
        | | | | | |
end | | | |
(1 row)
</pre></td></tr>
</table>

Configuration Item	Behavior
collection_exception_backcompat	Exception value thrown when an error related to the collection type is reported in the PL/SQL. Currently, three exception values are controlled. The mapping is as follows: Parameter not enabled Parameter enabled collection_is_null program_limit_exceeded subscript_beyond_count program_limit_exceeded subscript_outside_limit program_limit_exceeded Example: gaussdb=# create or replace procedure p1 is gaussdb$# type t1 is table of int; gaussdb$# v t1; gaussdb$# v_int int; gaussdb$# begin gaussdb$# v_int := v.count(); gaussdb$# exception when collection_is_null then gaussdb$# raise info '%', sqlerrm; gaussdb$# end; gaussdb$# / CREATE PROCEDURE gaussdb=# call p1(); INFO: Reference to uninitialized collection p1 ---- (1 row) gaussdb=# create or replace procedure p1 is gaussdb$# type t1 is table of int; gaussdb$# v t1 := t1(1, 2, 3); gaussdb$# v_int int; gaussdb$# begin gaussdb$# v_int := v(4); gaussdb$# exception when subscript_beyond_count then gaussdb$# raise info '%', sqlerrm; gaussdb$# end; gaussdb$# / CREATE PROCEDURE gaussdb=# call p1(); INFO: Subscript beyond count p1 ---- (1 row) gaussdb=# create or replace procedure p1 is gaussdb$# type t1 is table of int; gaussdb$# v t1 := t1(1, 2, 3); gaussdb$# v_int int; gaussdb$# begin gaussdb$# v_int := v(-1); gaussdb$# exception when subscript_outside_limit then gaussdb$# raise info '%', sqlerrm; gaussdb$# end; gaussdb$# / CREATE PROCEDURE gaussdb=# call p1(); INFO: Subscript outside of limit p1

Configuration Item	Behavior
	---- (1 row) gaussdb=# set behavior_compat_options = 'collection_exception_backcompat'; SET gaussdb=# create or replace procedure p1 is gaussdb$# type t1 is table of int; gaussdb$# v t1; gaussdb$# v_int int; gaussdb$# begin gaussdb$# v_int := v.count(); gaussdb$# exception when program_limit_exceeded then gaussdb$# raise info '%', sqlerrm; gaussdb$# end; gaussdb$# / CREATE PROCEDURE gaussdb=# call p1(); INFO: Reference to uninitialized collection p1 ---- (1 row) gaussdb=# create or replace procedure p1 is gaussdb$# type t1 is table of int; gaussdb$# v t1 := t1(1, 2, 3); gaussdb$# v_int int; gaussdb$# begin gaussdb$# v_int := v(4); gaussdb$# exception when program_limit_exceeded then gaussdb$# raise info '%', sqlerrm; gaussdb$# end; gaussdb$# / CREATE PROCEDURE gaussdb=# call p1(); INFO: Subscript beyond count p1 ---- (1 row) gaussdb=# create or replace procedure p1 is gaussdb$# type t1 is table of int; gaussdb$# v t1 := t1(1, 2, 3); gaussdb$# v_int int; gaussdb$# begin gaussdb$# v_int := v(-1); gaussdb$# exception when program_limit_exceeded then gaussdb$# raise info '%', sqlerrm; gaussdb$# end; gaussdb$# / CREATE PROCEDURE gaussdb=# call p1(); INFO: Subscript outside of limit p1 ---- (1 row)

Configuration Item	Behavior
enable_case_when_alias	If this parameter is enabled, the aliases of the CASE WHEN and DECODE syntaxes are character strings starting with __unnamed_. Example: gaussdb=# set behavior_compat_options='enable_case_when_alias'; SET gaussdb=# create table test(c1 varchar2); CREATE TABLE gaussdb=# insert into test values('x'); INSERT 0 1 gaussdb=# select decode(c1,'x','0','default') from test; __unnamed_decode__ -------------------- 0 (1 row) gaussdb=# select (case c1 when 'x' then '0' else 'default' end) from test; __unnamed_case_when__ ---------------------- 0 (1 row)

Configuration Item	Behavior
plsql_rollback_keep_user	Determines whether ROLLBACK and ROLLBACK TO SAVEPOINT in PL/SQL changes the current user. If this parameter is enabled, ROLLBACK in the PL/SQL does not change the current user. Example: gaussdb=# create user plsql_rollback1 PASSWORD '********'; gaussdb=# create user plsql_rollback2 PASSWORD '********'; gaussdb=# grant plsql_rollback1 to plsql_rollback2; gaussdb=# create or replace procedure plsql_rollback1.p1 () authid definer gaussdb-# as gaussdb$# va int; gaussdb$# begin gaussdb$# raise info 'current usr:%', current_user; gaussdb$# rollback; gaussdb$# raise info 'current usr:%', current_user; gaussdb$# end; gaussdb$# / CREATE PROCEDURE gaussdb=# set session AUTHORIZATION plsql_rollback2 PASSWORD '********'; SET gaussdb=> set behavior_compat_options = 'plsql_rollback_keep_user'; SET gaussdb=> call plsql_rollback1.p1 (); INFO: current usr:plsql_rollback1 INFO: current usr:plsql_rollback1 p1 ---- (1 row) **CAUTION** • This parameter is valid only in an A-compatible database.

Configuration Item	Behavior
forbid_package_function_with_prefix	After this parameter is enabled, an error is reported if a function with a prefix is created in a package. For example: -- Create a schema **sch1**. gaussdb=# CREATE SCHEMA sch1; -- An error is reported after a package is created. gaussdb=# CREATE PACKAGE pck1 IS PROCEDURE sch1.pck1(); END pck1; / ERROR: not support procedure name use *.* format in package at or near ";" CONTEXT: compilation of PL/pgSQL package near line 1

Configuration Item	Behavior
enable_bpcharlikebpchar_compare	Enables or disables the bpcharlikebpchar and bpcharnlikebpchar operators. ● This parameter is enabled by default for a newly installed database. ● After the database of a version earlier than 505.1.0 is upgraded, this parameter is disabled by default. gaussdb=# SELECT bpcharlikebpchar('455'::BPCHAR(10), '455 '::BPCHAR); bpcharlikebpchar ------------------ f (1 row) gaussdb=# SELECT bpcharlikebpchar('455 '::BPCHAR(10), '455 '::BPCHAR(10)); bpcharlikebpchar ------------------ t (1 row) gaussdb=# SELECT bpcharlikebpchar('455 '::BPCHAR(10), '455'::BPCHAR(10)); bpcharlikebpchar ------------------ t (1 row) gaussdb=# SELECT bpcharlikebpchar('455 '::BPCHAR(10), '455'::BPCHAR(11)); bpcharlikebpchar ------------------ f (1 row) gaussdb=# CREATE TABLE op_test (col BPCHAR(2) DEFAULT NULL); CREATE TABLE gaussdb=# CREATE INDEX op_index ON op_test(col); CREATE INDEX gaussdb=# INSERT INTO op_test VALUES ('a'); INSERT 0 1 gaussdb=# INSERT INTO op_test VALUES ('1'); INSERT 0 1 gaussdb=# INSERT INTO op_test VALUES ('11'); INSERT 0 1 gaussdb=# INSERT INTO op_test VALUES ('12'); INSERT 0 1 gaussdb=# INSERT INTO op_test VALUES ('sd'); INSERT 0 1 gaussdb=# INSERT INTO op_test VALUES ('aa'); INSERT 0 1 gaussdb=# SHOW behavior_compat_options; behavior_compat_options ------------------------- (1 row) -- If **behavior_compat_options** does not contain **enable_bpcharlikebpchar_compare**, the latest bpcharlikebpchar operator is not enabled and the result set returned by the matching between bpchars is not the same as expected (all data should be returned in normal cases). gaussdb=# EXPLAIN (COSTS OFF) SELECT * FROM op_test WHERE col LIKE col::BPCHAR ORDER BY col; QUERY PLAN -------------------------------------- Sort Sort Key: col -> Seq Scan on op_test Filter: (col ~~ (col)::text) (4 rows) gaussdb=# SELECT * FROM op_test WHERE col LIKE col::BPCHAR ORDER BY

Configuration Item	Behavior
	col; col ----- 11 12 aa sd (4 rows) gaussdb=# SET behavior_compat_options = 'enable_bpcharlikebpchar_compare'; SET gaussdb=# SHOW behavior_compat_options; behavior_compat_options --------------------------------- enable_bpcharlikebpchar_compare (1 row) -- After this parameter is enabled, the latest bpcharlikebpchar operator is enabled, and the returned behavior meets the expected behavior during matching. gaussdb=# EXPLAIN (COSTS OFF) SELECT * FROM op_test WHERE col LIKE col::BPCHAR ORDER BY col; QUERY PLAN ------------------------------ Sort Sort Key: col -> Seq Scan on op_test Filter: (col ~~ col) (4 rows) gaussdb=# SELECT * FROM op_test WHERE col LIKE col::BPCHAR ORDER BY col; col ----- 1 11 12 a aa sd (6 rows) gaussdb=# DROP TABLE op_test; DROP TABLE **NOTE** • This parameter takes effect only when **sql_compatibility** is set to **A**. • If this parameter is enabled, the result set and execution plan for LIKE or NOT LIKE pattern matching between BPCHAR types are affected.
cursor_asensitive	Used for forward compatibility of cursor data-sensitive behavior. This item is not supported in the centralized mode.

Configuration Item	Behavior
show_full_error_lineno	Displays the number of the line that contains the header information of a stored procedure. When compilation error information is printed during the creation of a stored procedure, the number of the line where the CREATE statement starts is displayed. When a statement with compilation problems is called, there are two cases: if the recompilation is non-invalid, the number of the line where the CREATE statement starts is displayed; if an error is reported indicating invalid recompilation, the line number is displayed based on the original logic. To display the error information, you must have permission to view gs_source which contains the original definition of the stored procedure; otherwise, the number of the line where the error is reported is displayed based on the original logic. If an error is reported when the stored procedure is executed, after this parameter is set, the number of the line where the CREATE statement starts is displayed. This parameter takes effect only in centralized mode. set behavior_compat_options='plpgsql_dependency, show_full_error_lineno'; set plsql_compile_check_options='plsql_expression_check'; CREATE OR REPLACE function proclex1(type1 int) return int is var1 int; var2 int; begin var3 := 30; --- **var3** is not defined. return type1; end; / WARNING: "var3" is not a known variable DETAIL: N/A CONTEXT: compilation of PL/pgSQL function "proclex1" near line 6 WARNING: Function created with compilation errors.

Configuration Item	Behavior
enable_crosstype_integer_operator	Enables or disables the cross-type integer operator. ● This parameter is enabled by default for a newly installed database. ● After the database of a version earlier than 505.1.0 is upgraded, this parameter is disabled by default. ● Involved operators: =, <>, <, >, <=, and >= ● After this parameter is enabled, the involved cross-type integers can be compared directly without implicit conversion. Here is an example that uses (int1 op int2): gaussdb=# CREATE TABLE implicit_index(c1 int1); CREATE TABLE gaussdb=# CREATE INDEX idx1 ON implicit_index(c1); CREATE INDEX gaussdb=# SET behavior_compat_options='enable_crosstype_integer_operator'; SET gaussdb=# EXPLAIN SELECT * FROM implicit_index WHERE c1 = 1::int2; QUERY PLAN -- [Bypass] Index Only Scan using idx1 on implicit_index (cost=0.00..4.48 rows=13 width=1) Index Cond: (c1 = 1::smallint) (3 rows) gaussdb=# SET behavior_compat_options=''; SET gaussdb=# EXPLAIN SELECT * FROM implicit_index WHERE c1 = 1::int2; QUERY PLAN -- Seq Scan on implicit_index (cost=0.00..49.52 rows=13 width=1) Filter: ((c1)::bigint = 1::smallint) (2 rows) gaussdb=# DROP TABLE implicit_index; DROP TABLE

Configuration Item	Behavior
	NOTE • This parameter takes effect only when **sql_compatibility** is set to **A**. • If this parameter is enabled, the execution plan of operators involved in the following scenarios will be affected: • (int1 op int2) • (int1 op int4) • (int1 op int8) • (int1 op int16) • (int1 op numeric) • (int2 op int1) • (int2 op int16) • (int2 op numeric) • (int4 op int1) • (in4 op int16) • (int4 op numeric) • (int8 op int1) • (int8 op int16) • (int8 op numeric) • (int16 op int1) • (int16 op int2) • (int16 op int4) • (int16 op int8) • (numeric op int1) • (numeric op int2) • (numeric op int4) • (numeric op int8)
time_constexpr_compact	If this parameter is enabled, in a scenario where the time expression is executed, the type automatically returned is with timezone or without timezone depending on whether the constant carries the time zone. Currently, the timestamp and time types are supported. `-- Execute a timestamp expression without a time zone.` `gaussdb=# SELECT timestamp '1999-03-15 8:00:00';` `     timestamp` `---------------------` ` 1999-03-15 08:00:00` `(1 row)` `-- The type timestamp with time zone is returned after a timestamp expression with a time zone is executed.` `gaussdb=# SELECT timestamp '1999-03-15 8:00:00 -8:00:00';` `     timestamptz` `------------------------` ` 1999-03-16 00:00:00+08` `(1 row)`

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Adjust the parameter value based on the database-compatible objects.

plsql_compile_check_options

Parameter description: Specifies the database compatibility configuration item.

Parameter type: string.

Unit: none

Value range: Currently, only compatibility configuration items listed in **Table 14-15** are supported. Use commas (,) to separate multiple compatibility configuration items, for example, **set plsql_compile_check_options='for_loop,outparam'**.

Default value: **''**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none.

Table 14-15 Compatibility configuration items

Configuration Item	Behavior
for_loop	Controls the behavior of the FOR_LOOP query statement in a stored procedure. When this parameter is set, if *rec* has been defined in the FOR rec IN query LOOP statement, the defined *rec* variable is not reused. Instead, a new variable is created. Otherwise, the defined *rec* variable is reused and no variable is created. (It is the same as **proc_implicit_for_loop_variable** and will be incorporated later.)
outparam	When the output parameter overloading condition is met, the output parameters are checked. If the output parameters are constant, an error is reported.

Configuration Item	Behavior
plsql_expression_check	If the **plsql_expression_check** parameter is enabled, the **plpgsql_dependency** parameter must also be enabled. It verifies full statements in functions, stored procedures, and packages. If an undefined object exists (for example, a table, variable, or function does not exist), an alarm is generated. **CAUTION** • The verification function supports only the following SQL statements: SELECT, INSERT, UPDATE, DELETE, and MERGE. DDL and DML statements are not supported. • Dynamic SQL statements are not verified. • False positives exist. As a result, alarms are generated when functions, stored procedures, and packages are created, invalid recompilation occurs, or the **valid** column of pg_object is **false**. Therefore, you are advised to use it in the test environment to debug functions, stored procedures, and packages and modify the definitions of functions, stored procedures, and packages to improve fault locating efficiency. It is not recommended in the production environment. The following is an example of the verification function: `CREATE OR REPLACE procedure proc1()` `AS` `var1 int;` `begin` `SELECT id into var1 from table1_not_exist limit 1;-- The table does not exist, and an alarm is generated.` `SELECT id into var1 from view1_not_exist limit 1;-- The view does not exist, and an alarm is generated.` `var1 := func_input1(func1_not_exist());-- The function does not exist, and an alarm is generated.` `var1 := pkg.var1_not_exist;-- The package variable does not exist, and an alarm is generated.` `end;` `/` The following command output indicates that the creation is successful and the alarm information is printed: `WARNING: compile failed when parse the query: select id from table1_not_exist limit 1, error info is relation "table1_not_exist" does not exist on datanode` `DETAIL: N/A` `CONTEXT: compilation of PL/SQL function "proc1" near line 3` `WARNING: compile failed when parse the query: select id from view1_not_exist limit 1, error info is relation "view1_not_exist" does not exist on datanode` `DETAIL: N/A` `CONTEXT: compilation of PL/SQL function "proc1" near line 4` `WARNING: compile failed when parse the query: SELECT func_input1(func1_not_exist()), error info is function func1_not_exist() does not exist` `DETAIL: N/A` `CONTEXT: compilation of PL/SQL function "proc1" near line 5` `WARNING: compile failed when parse the query: SELECT pkg.var1_not_exist, error info is missing FROM-clause entry for table "pkg"` `DETAIL: N/A` `CONTEXT: compilation of PL/SQL function "proc1" near line 6` `WARNING: Procedure created with compilation errors.` `CREATE PROCEDURE` False alarms:

Configuration Item	Behavior
	Example 1: CREATE TABLE emp(deptno smallint, ename char(100), salary int); CREATE TABLE emp_back(deptno smallint, ename char(100), salary int); CREATE OR REPLACE PROCEDURE proc_200() As BEGIN for data in delete from emp returning * loop INSERT INTO emp_back values(data.deptno,data.ename,data.salary);-- This alarm is falsely reported. END LOOP; END; / WARNING: compile failed when parse the query: INSERT INTO emp_back values(data.deptno,data.ename,data.salary), error info is record "data" is not assigned yet when get datum type info DETAIL: N/A CONTEXT: compilation of PL/pgSQL function "proc_200" near line 4 WARNING: Procedure created with compilation errors. CREATE PROCEDURE Example 2: CREATE OR REPLACE procedure proc1() is BEGIN DROP TABLE if exists table1; CREATE TABLE table1(id int); INSERT INTO table1 values(1);-- The system incorrectly reports that **table1** does not exist. END; / WARNING: compile failed when parse the query: insert into table1 values(1), error info is relation "table1" does not exist on datanode DETAIL: N/A CONTEXT: compilation of PL/SQL function "proc1" near line 2 WARNING: Procedure created with compilation errors. CREATE PROCEDURE The status of pg_object is invalid. SELECT object_type,valid from pg_object obj join pg_proc proc on obj.object_oid = proc.oid and proc.proname = 'proc1'; object_type | valid -------------+------- P | f (1 row) ALTER procedure proc1 compile; WARNING: compile failed when parse the query: insert into table1 values(1), error info is relation "table1" does not exist on datanode DETAIL: N/A CONTEXT: compilation of PL/SQL function "proc1" near line 5 WARNING: Procedure proc1 recompile with compilation errors. ALTER PROCEDURE SELECT object_type,valid from pg_object obj join pg_proc proc on obj.object_oid = proc.oid and proc.proname = 'proc1'; object_type | valid -------------+------- P | f (1 row) call proc1(); NOTICE: table "table1" does not exist, skipping CONTEXT: SQL statement "drop table if exists table1" PL/SQL function proc1() line 3 at SQL statement proc1 ------- (1 row)

Configuration Item	Behavior
	SELECT object_type,valid from pg_object obj join pg_proc proc on obj.object_oid = proc.oid and proc.proname = 'proc1'; object_type | valid -------------+------- P | t (1 row) When a function, stored procedure, or package is created or recompiled, the verification function is executed. If this alarm is falsely reported, the **valid** column of pg_object changes to **false**. In the execution phase, no verification is performed. When the execution is successful, the **valid** column of pg_object changes to **true**.

a_format_version

Parameter description: Specifies the database platform compatibility configuration item. The value of this parameter is an enumerated string.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string.

Default value: ""

NOTE

- Currently, only **Table 14-16** is supported.
- Set a character string for the compatibility configuration item, for example, **set a_format_version='10c'**.

Table 14-16 Compatibility configuration items

Configuration Item	Behavior
10c	Compatible version of platform A

a_format_dev_version

Parameter description: Specifies the database platform minor version compatibility configuration item. The value of this parameter is an enumerated string.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string.

Default value: ""

NOTE

- Currently, only **Table 14-17** is supported.
- Set a character string for the compatibility configuration item, for example, **set a_format_dev_version='s1'**.

Table 14-17 Compatibility configuration items

Configuration Item	Behavior
s1	• Compatible minor version of platform A, which affects functions TRUNC(date, fmt), ROUND(date, fmt), NVL2, LPAD, RPAD, ADD_MONTHS, MONTHS_BETWEEN, REGEXP_REPLACE, REGEXP_COUNT, TREAT, EMPTY_CLOB, INSTRB, trunc(number), greatest, least, mod, round(number), cast, to_date, to_timestamp, chr, rtrim, translate, to_char, to_number, and to_timestamp_tz. • Data type casting: A decimal character string is rounded off when it is converted to an integer (int1/int2/int4/int8/int16). • Data type casting: Implicit conversion from timestamp with time zone to timestamp without time zone is supported.
s2	• Compatible minor version of platform A, which affects functions such as dump, to_single_byte, to_multi_byte, nls_upper, nls_lower, initcap, ascii2, asciistr, unistr, vsize, cosh, remainder, sinh, tanh, nanvl, current_date, current_timestamp, dbtimezone, numtodsinterval, numtoyminterval, new_time, sessiontimezone, sys_extract_utc, tz_offset, to_binary_double, to_binary_float, to_dsinterval, to_yminterval, lnnvl, ora_hash, rawtohex2, bit2coding, and bit4coding. • Supports all behaviors when the compatibility configuration item is set to **s1**.
s3	• Compatible minor version of platform A. If the parameter is enabled, nested calling of functions without parameters is supported. • Supports all behaviors when the compatibility configuration item is set to **s2**.
s4	• Compatible minor version of platform A, which affects functions such as nchr(cvalue int|bigint), to_timestamp_tz, getclobval(xml), and getstringval(xml). • Supports all behaviors when the compatibility configuration item is set to **s3**.
s5	• Compatible minor version of platform A. If the parameter is enabled, the composite type with the same name as a function is preferentially parsed. The sys_guid() function is affected. • Supports all behaviors when the compatibility configuration item is set to **s4**.

b_format_version

Parameter description: Specifies the database platform compatibility behavior configuration item that controls the forward compatibility in B-compatible mode.

NOTE

- **b_format_version** takes effect only when **sql_compatibility** is set to **B**.
- If this parameter is set to a non-empty string, **b_format_behavior_compat_options** is set to **'ALL'** and **bytea_output** parameter is set to **'escape'** simultaneously. If this parameter is set to an empty string and takes effect, the **b_format_behavior_compat_options** and **bytea_output** are set to the original values.

Parameter type: string.

Unit: none

Value range: an empty string **''** or **'5.7'**

Default value: an empty string **''**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 1 GUC parameters**.

Setting suggestion: none.

b_format_dev_version

Parameter description: Specifies the database platform minor version compatibility configuration item.

NOTE

b_format_dev_version takes effect only when **sql_compatibility** is set to **B** and **b_format_version** is set to **5.7**.

Parameter type: string.

Unit: none

Value range: an empty string **''** or a compatibility configuration item in **Table 14-18**.

Table 14-18 Compatibility configuration items

Configuration Item	Behavior
s1	1. The following parameters are influenced: If this parameter is enabled, the **datetime** option in the **disable_keyword_options** parameter is removed, indicating that DATETIME is used as a non-reserved keyword. *datetime* can be used as a table name, column name, or alias, but cannot be used as a function name, stored procedure name, or parameter name of a function or stored procedure. 2. The following functions are influenced: curdate, from_days, date_format, str_to_date, current_date, datediff, timestampdiff, date_add, subtime, month, time_to_sec, to_days, to_seconds, dayname, monthname, convert_tz, sec_to_time, addtime, adddate, date_sub, timediff, last_day, weekday, from_unixtime, unix_timestamp, subdate, day, year, weekofyear, dayofmonth, dayofyear, week, yearweek, dayofweek, time_format, hour, minute, second, microsecond, quarter, utc_date, get_format, extract, makedate, period_add, timestamppadd, period_diff, utc_time, utc_timestamp, sysdate, current_timestamp, maketime, curtime, current_time, localtime, localtimestamp, now, lc_time_names, default_week_format, and json_object. 3. The following types are influenced: a. Be compatible with the integer type tinyint with the value range changed to –128 to 127. b. Be compatible with the character string types char and varchar. The precision *n* of char(*n*) and varchar(*n*) is changed from the byte length to the character length. c. Be compatible with the text types tinytext, mediumtext, and longtext. These types are mapped to the text type after the parameter is enabled. d. Be compatible with binary types tinyblob, blob, mediumblob, and longblob. These types are mapped to the bytea type after the parameter is enabled. e. Be compatible with floating-point types double, float, and real. Among them, double is mapped to the float8 type, double(p, s) is mapped to the numeric type, float(p, s) is mapped to the numeric type, real is mapped to float8, and real(p, s) is mapped to the numeric type. f. Be compatible with high-precision types numeric, dec, and decimal. If the precision and scale are not specified, the default precision is changed to **10** and the scale is changed to **0**.

Configuration Item	Behavior
	g. Be compatible with time types datetime, timestamp, time, and date. Replace datetime[(*p*)] with a timestamp without time zone[(*p*)]. Replace timestamp[(p)] with a timestamp with time zone[(p)]. The input, output, range, and precision specifications of these types are changed. For details, see "SQL Reference > Data Type" in *Developer Guide*. h. Be compatible with integer types tinyint, smallint, int, and bigint. The display width and ZEROFILL attributes of these types take effect after the parameter is enabled. 4. The following syntax is influenced: The column constraint ON UPDATE update_expr can be specified. 5. The following operators are influenced: a. An error is reported when the || operator between bytea types is used. b. The REGEXP operator is supported.

Configuration Item	Behavior
s2	The compatibility behavior controlled by s1 is included. In addition, the following impacts are included: 1. The following parameters are influenced: a. The **auto_increment_cache** parameter takes effect only when this option is enabled. b. When this option is enabled, the **multi_insert_min_rows** parameter takes effect for auto-increment columns. c. When this option is enabled, **standard_conforming_strings** and **escape_string_warning** are set to **off**. d. The **collation_connection** and **character_set_connection** parameters take effect only when this option is enabled. 2. The following functions are affected: a. When **0**, **NULL**, and definite values are imported or batch inserted into the auto-increment column, the auto-increment count is updated immediately when the definite values are inserted, and then **0** or **NULL** increases automatically based on the definite values. b. The like operator does not report an error when an escape character is at the end of the matching string. c. The sorting rule priorities of character sets and collations will be changed. For details, see section 7.4.7 in *Developer Guide (Centralized)*. d. After the parameter is enabled, the collation of foreign keys cannot be inconsistent with that of columns. 3. The following syntax is influenced: a. The CREATE TABLE table_name LIKE source_table syntax is supported. b. The syntaxes CREATE TABLE table_name LIKE source_table and CREATE TABLE table_name (LIKE source_table) cannot specify **INCLUDING** and **EXCLUDING** options. By default, **INCLUDING ALL** is specified. c. The LOAD DATA syntax is supported. Some syntax functions in gs_loader that are consistent with the LOAD DATA syntax will change. d. The collate clause can be specified by set names. e. The syntaxes for changing table names, such as ALTER TABLE and RENAME TABLE, have impact in the following scenario:

Configuration Item	Behavior
	If the character string corresponding to the new table name starts with "#MySQL50#" and is followed by other characters, "#MySQL50#" will be ignored. If the old and new table names are the same, no error is reported.

Default value: an empty string **''**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 1 GUC parameters**.

Setting suggestion: none.

m_format_dev_version

Parameter description: Specifies the database platform minor version compatibility configuration item.

NOTE

m_format_dev_version takes effect only when **sql_compatibility** is set to **M**.

Parameter type: string.

Unit: none

Value range: an empty string **''** or a compatibility configuration item in **Table 14-19**.

Table 14-19 Compatibility configuration items

Configuration Item	Behavior
s1	The following syntax is influenced: 1. Disable the SELECT FETCH FIRST WORS ONLY syntax. 2. Disable the TRUNCATE CASCADE/RESTRICT syntax. 3. USING INDEX TABLESPACE tablespace_name syntax cannot be specified when a primary key is defined using CREATE TABLE. 4. The semantics of CASCADE/RESTRIC syntax in DROP TABLE/VIEW/COLUMN is ignored, and the default behavior is RESTRIC. 5. If the CREATE TABLE REFERENCES syntax is used as a column constraint, its semantics is changed to ignore the foreign key definition.

Default value: an empty string **''**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 1 GUC parameters**.

Setting suggestion: none.

sql_mode

Parameter description: Specifies the SQL behavior control configuration item in B-compatible and M-compatible modes.

NOTE

sql_mode takes effect when **sql_compatibility** is set to **B**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s1'**, or when **sql_mode** takes effect only when **sql_compatibility** is set to **M**.

Parameter type: string.

Unit: none

Value range:

An empty string **''** or a value in the range of **'strict_trans_tables,only_full_group_by,no_zero_in_date,no_zero_date,error_for_division_by_zero,no_auto_create_user,no_engine_substitution,pad_char_to_full_length,no_auto_value_on_zero'** in B-compatible mode.

In M-compatible mode, the parameters supported in B-compatible mode, **no_unsigned_subtraction**, **ansi_quotes**, **allow_invalid_dates**, and **real_as_float** are supported.

The meaning of each option value is described in **Table 14-20**.

Table 14-20 Compatibility configuration items

Configuration Item	Behavior	B-compatible Parameter Configuration	M-compatible Parameter Configuration
strict_trans_tables	Currently, only data types and system functions in B-compatible and M-compatible databases can be verified. If this parameter is set, its format and range are strictly verified. If an invalid value is entered or the value exceeds the range, an error is reported during parsing. If this parameter is not set, the format and range of the input parameter are loosely verified. If an invalid value is entered or the value exceeds the range, a warning is reported during parsing and the value **0** of the corresponding data is returned.	The options **strict_trans_tables**, **only_full_group_by**, **no_zero_in_date**, **no_zero_date**, and **error_for_division_by_zero** can only be set or canceled at the same time.	It can be set and canceled independently.
only_full_group_by	Projection columns that contain non-GROUP BY keys and that are not constants and aggregate functions are not allowed.		It can be set and canceled independently.
no_zero_in_date	The year, month, and day of the DATE cannot be **0**.		It can be set and canceled independently.
no_zero_date	The value of DATE cannot be **0** (**0000-00-00**).		It can be set and canceled independently.
error_for_division_by_zero	The value cannot be divided by 0.		It can be set and canceled independently.
no_auto_create_user	This item has no actual function. It is used only for compatibility that an error is not reported when the SET SQL_MODE statement contains this option.	It can be set and canceled independently.	It can be set and canceled independently.

Configuration Item	Behavior	B-compatible Parameter Configuration	M-compatible Parameter Configuration
no_engine_substitution	This item has no actual function. It is used only for compatibility that an error is not reported when the SET SQL_MODE statement contains this option.	It can be set and canceled independently.	It can be set and canceled independently.
pad_char_to_full_length	Be used for formatted output of CHAR columns in a table. If there are CHAR columns, a string with spaces at the end is output. Otherwise, a string without spaces at the end is output.	It can be set and canceled independently.	It can be set and canceled independently.
no_auto_value_on_zero	If this option is included, the value **0** inserted into the **AUTO_INCREMENT** column does not trigger auto-increment.	It can be set and canceled independently.	It can be set and canceled independently.
no_unsigned_subtraction	Unsigned numbers cannot be subtracted. If unsigned integers are subtracted and the result is negative, an error is returned.	-	It can be set and canceled independently.
ansi_quotes	ANSI_QUOTES mode is enabled. In this mode, double quotation marks are treated as identifier quotes rather than string quotes, and you must quote the entire name of a table or column with double quotation marks instead of backquotes.	-	It can be set and canceled independently.
allow_invalid_dates	If this option is enabled, MySQL allows invalid dates to be inserted, for example, '0000-00-00'.	-	It can be set and canceled independently.
real_as_float	By default, the REAL type is regarded as the DOUBLE type. If this option is enabled, the REAL type is regarded as the FLOAT type.	-	It can be set and canceled independently.

Default value:
'strict_trans_tables,only_full_group_by,no_zero_in_date,no_zero_date,error_for_division_by_zero,no_auto_create_user,no_engine_substitution'

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none.

auto_increment_increment

Parameter description: Specifies the auto-increment step of an auto-increment column. The auto-increment value is calculated by the following formula: **auto_increment_offset** + *N* × **auto_increment_increment**. *N* is a positive integer. If the parameter value is smaller than that of **auto_increment_offset**, an error occurs when the values in the auto-increment column automatically increase.

Parameter type: integer.

Unit: none

Value range: 1 to 65535

Default value: **1**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none.

auto_increment_offset

Parameter description: Specifies the initial value of an auto-increment column. The auto-increment value is calculated by the following formula: **auto_increment_offset** + *N* × **auto_increment_increment**. *N* is a positive integer. If the parameter value is greater than that of **auto_increment_increment**, an error occurs when the values in the auto-increment column automatically increase.

Parameter type: integer.

Unit: none

Value range: 1 to 65535

Default value: **1**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none.

auto_increment_cache

Parameter description: Specifies the number of reserved auto-increment cache values when auto-increment is triggered by batch insertion or import of auto-increment columns. When auto-increment values are reserved, the auto-increment counter value is updated to the maximum auto-increment cache value. Before the cache values are used up, the auto-increment counter value remains unchanged, and the triggered auto-increment uses the cache values.

NOTE

- The reserved cache values are valid only in the statement. If the reserved auto-increment cache values are used up and subsequent INSERT statements trigger auto-increment based on the auto-increment counter, the values in the auto-increment column in the table are discontinuous.
- When auto-increment is triggered by parallel import or insertion of auto-increment columns, the cache value reserved for each parallel thread is used only in the thread. If the cache value is not used up, the values of auto-increment columns in the table are discontinuous.
- When you add an auto-increment column to a table with data or modify a column to an auto-increment column, the existing data triggers auto-increment. The reserved auto-increment cache value is also affected by this parameter.
- This parameter does not affect the auto-increment column in the local temporary table.
- This parameter is valid only when **b_format_version** is set to **'5.7'** and **b_format_dev_version** is set to **'s2'**.

Default value: **0**

Parameter type: integer.

Unit: none

Value range: 0 to *INT_MAX*

- If this parameter is set to **0**, the auto-increment cache values are automatically reserved.
  - When auto-increment is triggered for the first time, if the number of rows to be inserted into the auto-increment column is known, the number is the reserved value.

    For example, the number of rows to be inserted cannot be obtained as the value of auto-increment that is triggered by INSERT INTO table SELECT ... or COPY FROM. When the ALTER TABLE statement is executed to rewrite table data, if auto-increment is triggered, reltuples in the statistics is used as the number of rows to be reserved. INSERT INTO table VALUES(...),(...),... is distributed to different DNs. Therefore, in some execution plans, DNs cannot obtain the number of rows to be inserted.
  - If the number of rows is unknown, 2^n values are reserved each time. For example, one value is reserved in the first auto-increment, two values are reserved in the second auto-increment, four values are reserved in the third auto-increment, and eight values are reserved for in fourth auto-increment. However, if the number of reserved values exceeds 65535, 65535 values are reserved.
- If this parameter is not set to **0**, the number of reserved cache values is the value of this parameter.
  - When auto-increment is triggered for the first time, if the number of rows to be inserted into the auto-increment column is known, the number is the reserved value.
  - If the number of rows is unknown, the value of **auto_increment_cache** is the number of auto-increment values reserved each time.

Default value: **0**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: The default value is recommended. However, if the auto-increment values are discontinuous in this case, you can adjust the parameter value based on the amount of data to be inserted in batches. Setting this parameter to a larger value improves the batch insertion performance but the auto-increment values are more likely to be discontinuous.

disable_keyword_options

Parameter description: Specifies database compatibility behavior. Multiple items are separated by commas (,). An identifier with this parameter set will not be used as a keyword.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range (non-M-compatible mode): a string. The following keywords are supported: auto_increment, change, charset, columns, compile, completion, containing, convert, csn, datetime, db4aishot, dbtimezone, discard_path, distributed, dumpfile, ends, entityescaping, escaped, evalname, event, events, expdp, extend, gsiusable, gsivalid, gsiwaitall, finish, ilm_pidx_list, impdp, ifnull, invisible, json_object, lines, link, lnnvl, load_discard, locked, mark, nocache, noentityescaping, noextend, noscale, nvl2, ordinality, outfile, performance, pivot, public, recover, regexp_like, scale, schedule, separator, sessiontimezone, shrink, slave, specification, starting, starts, subpartitions, substr, unpivot, verify, visible, wellformed, xmltype, regexp, rlike, and zerofill.

Default value (non-M-compatible mode): **"datetime,regexp,rlike,zerofill"**.

Value range (M-compatible mode): a string. The following keywords are supported: active, admin, array, authentication, buckets, bulk, challenge_response, clone, component, cume_dist, definition, dense_rank, description, empty, enforced, engine_attribute, except, exclude, factor, failed_login_attempts, finish, first_value, following, generate, geomcollection, get_master_public_key, get_source_public_key, grouping, groups, gtid_only, histogram, history, inactive, initial, initiate, intersect, invisible, json_table, json_value, keyring, lag, last_value, lateral, lead, locked, master_compression_algorithms, master_public_key_path, master_tls_ciphersuites, master_zstd_compression_level, member, nested, nowait, nth_value, ntile, nulls, of, off, oj, old, optional, ordinality, organization, others, over, password_lock_time, path, percent_rank, persist, persist_only, preceding, privilege_checks_user, process, random, rank, recursive, reference, registration, replica, replicas, require_row_format, resource, respect, restart, retain, returning, reuse, role, row_number, secondary, secondary_engine, secondary_engine_attribute, secondary_load, secondary_unload, skip, source_auto_position, source_bind, source_compression_algorithms, source_connect_retry, source_delay, source_heartbeat_period, source_host, source_log_file, source_log_pos, source_password, source_port, source_public_key_path, source_retry_count, source_ssl, source_ssl_ca, source_ssl_capath, source_ssl_cert, source_ssl_cipher, source_ssl_crl, source_ssl_crlpath, source_ssl_key, source_ssl_verify_server_cert, source_tls_ciphersuites, source_tls_version, source_user, source_zstd_compression_level, srid, stream, system, thread_priority, ties, tls, unbounded, unregister, url, vcpu, visible, window, zone, lc_collate, least, load_bad, load_discard, location, minvalue, move, nocycle, node, nomaxvalue, nominvalue, nvl, nvl2, oids, operator, owned, prepared, recyclebin, reindex, reject, relative, scroll, sequence, setof, shippable, size, slice, smalldatetime, smalldatetime_format, split, stable, stdin, stdout, strict, substring, sysdate, trim, unusable, vacuum, valid,

varchar2, verbose, version, within, xmlattributes, xmlconcat, xmlelement, xmlforest, xmlpi, xmlroot, and xmltype.

Default value (M-compatible mode): **""**

NOTE

If this parameter is enabled, some functions used as keywords will become invalid. Exercise caution when setting this parameter.

disable_plsql_keyword_options

Parameter description: Specifies the database compatibility behavior, that is, whether to use an identifier as a non-keyword. Values of this parameter are separated by commas (,).

Parameter type: string.

Unit: none

Value range:

- PIPE
- PIPELINED
- RANGE
- REPLACE
- SUBTYPE
- ''

Default value: ''

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none

NOTE

- If this parameter is enabled, some functions used as keywords will become invalid. Exercise caution when setting this parameter.
- To cancel the function of shielding the plsql keyword, leave this parameter empty.

plpgsql.variable_conflict

Parameter description: Sets the priority of using stored procedure variables and table columns with the same name.

This is a USERSET parameter. Set it based on method 3 provided in **Table 14-1**.

Value range: a string.

- **error** indicates that a compilation error is reported when the name of a stored procedure variable is the same as that of a table column.
- **use_variable** indicates that if the name of a stored procedure variable is the same as that of a table column, the variable is used preferentially.
- **use_column** indicates that if the name of a stored procedure variable is the same as that of a table column, the column name is used preferentially.

Default value: **error**

td_compatible_truncation

Parameter description: Specifies whether to enable features compatible with a Teradata database. You can set this parameter to **on** when connecting to a database compatible with the Teradata database, so that when you perform the **INSERT** operation, overlong strings are truncated based on the allowed maximum length before being inserted into char- and varchar-type columns in the target table. This ensures all data is inserted into the target table without errors reported.

NOTE

The string truncation function cannot be used if the **INSERT** statement includes a foreign table.

If inserting multi-byte character data (such as Chinese characters) to database with the character set byte encoding (such as SQL_ASCII or LATIN1), and the character data crosses the truncation position, the string is truncated based on its bytes instead of characters. Unexpected result will occur in tail after the truncation. If you want correct truncation result, you are advised to adopt encoding set such as UTF8, which has no character data crossing the truncation position.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that overlong strings are truncated.
- **off** indicates that overlong strings are not truncated.

Default value: **off**

uppercase_attribute_name

Parameter description: Specifies whether to return column names in uppercase to the client. This parameter is used only in the ORA-compatible mode and centralized environment.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that column names are returned to the client in uppercase.
- **off** indicates that column names are not returned to the client in uppercase.

Default value: **off**

character_set_connection

Parameter description: Specifies the character set of constant strings. If this parameter is modified, **collation_connection** is changed to the default collation of the character set.

- This parameter takes effect when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s2'**.
- This parameter takes effect in M-compatible mode.

Parameter type: character

Unit: none

Value range:

- Character sets supported when **sql_compatibility** is set to **'B'**. For details, see "COLLATE collation" in "SQL Reference > SQL Syntax > C > CREATE TABLE" in *Developer Guide*.
- Character sets within the value range supported in M-compatible mode. For details, see "SQL Reference > Character Set and Collation" in the *M-compatibility Development Guide*.
- Currently, the value cannot be different from that of the current database character set.

Default value: same as the value of **server_encoding**.

Setting method: This is a USERSET parameter, which can only be set using method 3, and cannot be set using the guc tools. For details about how to set this parameter, see **Table 14-1**.

Setting suggestion: none.

collation_connection

Parameter description: Specifies the collation of a constant string. If this parameter is modified, **character_set_connection** is changed to the default character set of the collation.

- This parameter takes effect when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s2'**.
- This parameter takes effect in M-compatible mode.

Parameter type: character

Unit: none

Value range:

- Character sets supported when **sql_compatibility** is set to **'B'**. For details, see "COLLATE collation" in "SQL Reference > SQL Syntax > C > CREATE TABLE" in *Developer Guide*.
- Character sets within the value range supported in M-compatible mode. For details, see "SQL Reference > Character Set and Collation" in the *M-compatibility Development Guide*.
- Currently, this parameter cannot be set to a value other than the collation corresponding to the current database character set.

Default value: default collation of the current **server_encoding**. If there is no default collation, the value is **'default'**.

Setting method: This is a USERSET parameter, which can only be set using method 3, and cannot be set using the guc tools. For details about how to set this parameter, see **Table 14-1**.

Setting suggestion: none.

character_set_results

Parameter description: Specifies the character set of the returned result.

- This parameter takes effect when **sql_compatibility** is set to **'B'**, **b_format_version** is set to **'5.7'**, and **b_format_dev_version** is set to **'s2'**.
- This parameter takes effect in M-compatible mode.

Parameter type: character

Unit: none

Value range:

- Character sets supported when **sql_compatibility** is set to **'B'**. For details, see "COLLATE collation" in "SQL Reference > SQL Syntax > C > CREATE TABLE" in *Developer Guide*.
- Character sets within the value range supported in M-compatible mode. For details, see "SQL Reference > Character Set and Collation" in the *M-compatibility Development Guide*.
- You can also set this parameter to **null** or **''**. In this case, the character set of the database is output.

Default value: same as the value of **server_encoding**.

Setting method: This is a USERSET parameter, which can only be set using method 3, and cannot be set using the guc tools. For details about how to set this parameter, see **Table 14-1**.

Setting suggestion: none.

lastval_supported

Parameter description: Specifies whether the lastval function can be used.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the lastval function can be used and the nextval function cannot be pushed down.
- **off** indicates that the lastval function cannot be used and the nextval function can be pushed down.

Default value: **off**

enable_copy_error_log

Parameter description: Enables or disables the error table **pgxc_copy_error_log**.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: indicates that the error table is enabled.
- **off**: indicates that the error table is disabled.

Default value: **off**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none.

loader_support_nul_character

Parameter description: Specifies the database platform minor version compatibility configuration item.

Parameter type: string.

Unit: none

Value range: '', **'s1'**, and **'s2'**. For details, see **Table 14-22**.

Default value: ''

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none.

NOTE

- This GUC parameter needs to be set when gs_loader is used to enable fault tolerance for nul characters (0x00).
- For details, see "Client Tools > gs_loader" in *Tool Reference*.

Table 14-21 Compatibility configuration items

Configuration Item	Behavior
s1	Specifies how gs_loader processes nul characters in data files during data import. If a data file contains the nul character, the data is truncated based on the position of the nul character. The data before the nul character is imported to the table, and the data after the nul character is truncated and discarded. This is used for forward compatibility of the function.
s2	Specifies how gs_loader processes nul characters in data files during data import. If a data file contains the nul character, the nul character is converted to the space character ' ' (0x20), and then processed, determined, and imported.
Empty	The default parameter settings do not affect any functions.

a_format_copy_version

Parameter description: Specifies the database platform minor version compatibility configuration item. The value of this parameter is an enumerated string.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**. When using **gs_loader** to import new features, you need to set the corresponding values.

Value range: a string.

Default value: ''

NOTE

- Currently, only **Table 14-22** is supported.
- Supports setting character string for compatibility configuration. For details, see "Client Tools > gs_loader" in *Tool Reference*. You can set the **a_format_copy_version** parameter by using **guc_param**.

Table 14-22 Compatibility configuration items

Configuration Item	Behavior
s1	• Supports gs_loader specifying data type (CHAR[(length)]/ INTEGER external[(length)]/FLOAT external[(length)]/ DECIMAL external[(length)]/TIMESTAMP/DATE/DATE EXTERNAL/INTEGER/SMALLINT/RAW[(length)]) to import data. • Supports gs_loader converting expressions and extending scenarios for columns.

enable_volatile_match_index

Parameter description: Specifies whether volatile functions can match indexes. This parameter is valid only when **DBCOMPATIBILITY** is set to **A**. There are semantic risks when the volatile function matches indexes. The stable and immutable functions can match indexes by default and comply with semantics. It is strongly recommended that you set this parameter based on the actual variability of the function instead of enabling this parameter.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **off**

enable_union_all_order

Parameter description: Specifies whether UNION ALL supports subquery order preserving when the main query is not sorted.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **on**

enable_immutable_optimization

Parameter description: Specifies whether immutable functions can be optimized. This parameter is valid only when **DBCOMPATIBILITY** is set to **A**. If a function with the immutable definition violates the immutable semantics, the call result, value, and impact on the caller are not defined.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: enabled.
- **off**: disabled.

Default value: **on**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-2**.

Setting suggestion: none.

enable_object_special_character

Parameter description: Determines whether the value of the **schema** parameter in the control file can contain any special characters in ["$'\] when the CREATE EXTENSION statement is executed and "@extschema@" is used in the script file.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-2**.

Value range: Boolean

- **on** indicates that the value can contain any special characters in ["$'\].
- **off** indicates that the value cannot contain any special characters in ["$'\].

Default value: **off**

NOTICE

The extended function is for internal use only. You are advised not to use it.

a_format_disable_func

Parameter description: Disables a specified system function. The value of this parameter consists of multiple system function OIDs separated by commas (,). The system function for which this parameter is set cannot be called.

When a system function cannot meet user requirements and a user-defined function with the same name is required, you can use this function.

Parameter type: string.

Unit: none

Value range: a string consisting of multiple system function OIDs separated by commas (,).

NOTE

This parameter can only be used to disable added system functions corresponding to the database platform compatibility behavior configuration items (**a_format_version** and **a_format_dev_version**). For details, see **Table 14-23**.

Table 14-23 System functions that can be disabled

Database Platform Compatibility Configuration Item	System Function That Can Be Disabled
10c,s1	anyarray array_extendnull(anyarray,int4,int4); -- **funcoid**: 6018 clob empty_clob(); -- **funcoid**: 3825 int4 instrb(text,text); -- **funcoid**: 3240 int4 instrb(text,text,int4); -- **funcoid**: 3241 int4 instrb(text,text,int4,int4); -- **funcoid**: 3242 numeric months_between(timestamp,timestamp); -- **funcoid**: 1522 timestamp round(timestamp); -- **funcoid**: obtains the OID by using the **select oid from pg_proc where proname='round' and pronamespace=11 and pronargs=1 and allargtypes[0]=1114** query statement. timestamp round(timestamp,text); -- **funcoid**: 4465 timestamp to_date(text,text,bool); -- **funcoid**: 1524 timestamp to_date(text,text,text,bool); -- **funcoid**: 1525 numeric to_number(text,text,bool); -- **funcoid**: 1787 numeric to_number(text,text,text,bool); -- **funcoid**: 1788 timestamp to_timestamp(text,text,bool); -- **funcoid**: 606 timestamp to_timestamp(text,text,text,bool); -- **funcoid**: 607 timestamptz to_timestamp_tz(text); -- **funcoid**: 1806 timestamptz to_timestamp_tz(text,text); -- **funcoid**: 1807 timestamptz to_timestamp_tz(text,text,bool); -- **funcoid**: 1808 timestamptz to_timestamp_tz(text,text,text,bool); -- **funcoid**: 1809

Database Platform Compatibility Configuration Item	System Function That Can Be Disabled
10c,s2	text DBTimezone(); -- **funcoid**: 5562 int8 ascii2(text); -- **funcoid**: 1625 text asciistr(text); -- **funcoid**: 1626 text asciistr(blob); -- **funcoid**: 1629 int4 bit2coding(text); -- **funcoid**: 9311 int4 bit4coding(text); -- **funcoid**: 9325 float8 cosh(float8); -- **funcoid**: 1548 numeric cosh(numeric); -- **funcoid**: 1549 timestamptz current_timestamp(numeric); -- **funcoid**: 3257 text dump(any); -- **funcoid**: 9086 text dump(any,int4); -- **funcoid**: 9088 text dump(any,int4,int4); -- **funcoid**: 9089 text dump(any,int4,int4,int4); -- **funcoid**: 9090 float4 nanvl(float4,float4); -- **funcoid**: 7112 float4 nanvl(float4,numeric); -- **funcoid**: 7115 float8 nanvl(float8,float8); -- **funcoid**: 7113 float4 nanvl(numeric,float4); -- **funcoid**: 7116 numeric nanvl(numeric,numeric); -- **funcoid**: 7114 timestamp new_time(timestamp,text,text); -- **funcoid**: 374 text nls_lower(text); -- **funcoid**: 4082 text nls_lower(text,text); -- **funcoid**: 4083 text nls_upper(text); -- **funcoid**: 4084 text nls_upper(text,text); -- **funcoid**: 4085 interval numtoyminterval(numeric,text); -- **funcoid**: 9333 int8 ora_hash(any); -- **funcoid**: 6127 int8 ora_hash(any,int8); -- **funcoid**: 6128 text rawtohex2(any); -- **funcoid**: 9540 numeric remainder(int8,int8); -- **funcoid**: 9767 numeric remainder(int2,int2); -- **funcoid**: 9765 numeric remainder(int4,int4); -- **funcoid**: 9766 float4 remainder(float4,float4); -- **funcoid**: 9771 float4 remainder(float4,numeric); -- **funcoid**: 9768 float8 remainder(float8,float8); -- **funcoid**: 9770 float4 remainder(numeric,float4); -- **funcoid**: 9769 numeric remainder(numeric,numeric); -- **funcoid**: 9761

Database Platform Compatibility Configuration Item	System Function That Can Be Disabled
	numeric remainder(int1,int1); -- **funcoid**: 9764 text session_time_zone(); -- **funcoid**: 9571 float8 sinh(float8); -- **funcoid**: 1546 numeric sinh(numeric); -- **funcoid**: 1547 timestamp sys_extract_utc(timestamp); -- **funcoid**: 5258 timestamp sys_extract_utc(timestamptz); -- **funcoid**: 5259 float8 tanh(float8); -- **funcoid**: 9762 numeric tanh(numeric); -- **funcoid**: 9763 float8 to_binary_double(text); -- **funcoid**: 9669 float8 to_binary_double(text,text); -- **funcoid**: 9670 float8 to_binary_double(text,text,bool); -- **funcoid**: 9671 float8 to_binary_double(text,text,text,bool); -- **funcoid**: 9672 float4 to_binary_float(text); -- **funcoid**: 9673 float4 to_binary_float(text,text); -- **funcoid**: 9674 float4 to_binary_float(text,text,bool); -- **funcoid**: 9675 float4 to_binary_float(text,text,text,bool); -- **funcoid**: 9676 blob to_blob(any); -- **funcoid**: 6990 interval to_dsinterval(text); -- **funcoid**: 9126 interval to_dsinterval(text,text,bool); -- **funcoid**: 9127 text to_multi_byte(text); -- **funcoid**: 9537 text to_multi_byte(blob); -- **funcoid**: 9539 varchar to_nchar(int8); -- **funcoid**: obtains the OID by using the **select oid from pg_proc where proname='to_nchar' and pronamespace=11 and pronargs=1 and allargtypes[0]=20** query statement. varchar to_nchar(int2); -- **funcoid**: obtains the OID by using the **select oid from pg_proc where proname='to_nchar' and pronamespace=11 and pronargs=1 and allargtypes[0]=21** query statement. varchar to_nchar(int4); -- **funcoid**: obtains the OID by using the **select oid from pg_proc where proname='to_nchar' and pronamespace=11 and pronargs=1 and allargtypes[0]=23;** query statement. text to_nchar(text); -- **funcoid**: obtains the OID by using the **select oid from pg_proc where proname='to_nchar' and pronamespace=11 and pronargs=1 and allargtypes[0]=25** query statement. varchar to_nchar(float4); -- **funcoid**: obtains the OID by using the **select oid from pg_proc where proname='to_nchar' and**

Database Platform Compatibility Configuration Item	System Function That Can Be Disabled
	pronamespace=11 and pronargs=1 and allargtypes[0]=700 query statement. varchar to_nchar(float8); -- **funcoid**: obtains the OID by using the **select oid from pg_proc where proname='to_nchar' and pronamespace=11 and pronargs=1 and allargtypes[0]=701** query statement. varchar to_nchar(numeric); -- **funcoid**: obtains the OID by using the **select oid from pg_proc where proname='to_nchar' and pronamespace=11 and pronargs=1 and allargtypes[0]=1700** query statement. text to_nchar(timestamp); -- **funcoid**: 5698 text to_nchar(timestamptz); -- **funcoid**: 5699 text to_nchar(anyset); -- **funcoid**: 5700 text to_nchar(int8,text); -- **funcoid**: 5694 text to_nchar(int4,text); -- **funcoid**: 5693 text to_nchar(float4,text); -- **funcoid**: 5695 text to_nchar(float8,text); -- **funcoid**: 5696 text to_nchar(timestamp,text); -- **funcoid**: 5697 text to_nchar(timestamptz,text); -- **funcoid**: 5691 text to_nchar(interval,text); -- **funcoid**: 5690 text to_nchar(numeric,text); -- **funcoid**: 5692 text to_single_byte(text); -- **funcoid**: 9536 text to_single_byte(blob); -- **funcoid**: 9538 interval to_yminterval(text); -- **funcoid**: 9124 interval to_yminterval(text,text,bool); -- **funcoid**: 9125 text tz_offset(text); -- **funcoid**: 706 text unistr(text); -- **funcoid**: 9081 text unistr(blob); -- **funcoid**: 9082 int4 vsize(any); -- **funcoid**: 9083
10c,s4	clob getclobval(xml); -- **funcoid**: 8011 varchar getstringval(xml); -- **funcoid**: 6976 nvarchar2 nchr(int8); -- **funcoid**: 1694 timestamptz to_timestamp_tz(text,text,text); -- **funcoid**: 1804 timestamptz to_timestamp_tz(text,text,text,text,bool); -- **funcoid**: 1805

Default value: ''

Setting method: This is a SUSET parameter. Set it based on instructions provided in **Table 14-2**.

Setting suggestion: If the database platform compatibility behavior configuration items (**a_format_version** and **a_format_dev_version**) are disabled, the corresponding added system functions are unavailable by default. You do not need to use this parameter to disable the functions.

enable_convert_illegal_char

Parameter description: Specifies whether invalid characters in the command output are not verified and are displayed as placeholders.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: The parameter is enabled. Special characters are replaced by the symbols specified by the **convert_illegal_char_mode** parameter during query.
- **off**: The parameter is disabled. If the query result contains characters that do not comply with the current character set encoding rule, an error is reported after verification.

Default value: **off**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-2**.

Setting suggestion: Retain the default value. Enable this parameter only when the data contains special characters and errors should not be reported for special characters.

NOTE

1. When the database character set is utf8, zhs16gbk, gb18030, gb18030-2022, or latin1, **enable_convert_illegal_char = on** takes effect. If the character set of the database client is different from that of the database server, invalid characters are displayed as placeholders.
2. Special character range: The special characters mentioned in this document include only fully abnormal encoding and hybrid encoding. The \u0000 character is not supported. If the character code contains the \u0000 character, the character is truncated at \u0000, which affects data integrity.
3. If the character sets of the database client and server are different, special characters beyond the character set of the server can be written to the database only by using the dbe_raw.cast_to_varchar2() function or the import and export tool.
4. When the GUC parameter is enabled, the behaviors of the special characters in the functions and advanced package functions listed in the following table are as follows:
   - When the character sets of the client and server are the same, no error is reported during the query of special characters. The behavior is the same as that before the GUC parameter is enabled.
   - If the character sets of the client and server are different, special characters are displayed as placeholders, which use question marks (?) by default.
   - You are advised not to use functions to process character strings that contain special characters. The functions listed in the following table do not report errors when processing character strings that contain special characters, and the correctness of the results cannot be ensured.

Table 14-24 Functions and advanced package functions supporting special characters

No.	Function Name/Advanced Package Function Name
1	bit_length(string)
2	btrim(string text [, characters text])
3	char_length(string) character_length(string)
4	chr(cvalue int|bigint) chr(integer)
5	concat(str1,str2)
6	concat_ws(sep text, str"any" [, str"any" [, ...]])
7	decode(string text, format text)
8	dump(expr[, return_fmt [, start_position [, length]]])
9	encode(data bytea, format text)
10	find_in_set(text, set)
11	format(formatstr text [, str"any" [, ...]])
12	left(str text, n int)
13	length(string)

No.	Function Name/Advanced Package Function Name
14	lengthb(text/bpchar)
15	ltrim(string [, characters])
16	md5(string)
17	notlike(x bytea name text, y bytea text)
18	octet_length(string)
19	overlay(string placing string FROM int [for int])
20	quote_ident(string text)
21	quote_literal(string text)
22	quote_nullable(string text)
23	rawcat(raw,raw)
24	regexp_count(string text, pattern text [, position int [, flags text]])
25	regexp_instr(string text, pattern text [, position int [, occurrence int [, return_opt int [, flags text]]]])
26	regexp_like(source_string, pattern [, match_parameter]) regexp_like(text,text,text)
27	regexp_matches(string text, pattern text [, flags text])
28	regexp_replace(string, pattern, replacement [,flags])
29	regexp_split_to_array(string text, pattern text [, flags text])
30	regexp_split_to_table(string text, pattern text [, flags text])
31	regexp_substr(source_char, pattern) regexp_substr(string text, pattern text [, position int [, occurrence int [, flags text]]])
32	repeat(string text, number int)
33	repexp_replace(string text, pattern text [, replacement text [, position int [, occurrence int [, flags text]]]])
34	replace(string text, from text, to text)
35	replace(string, substring)
36	reverse(str)
37	right(str text, n int)
38	rtrim(string [, characters])

No.	Function Name/Advanced Package Function Name
39	sha(string)
40	sha1(string)
41	sha2(string, hash_length)
42	split_part(string text, delimiter text, field int)
43	substring(string [from int] [for int]) substring(string from pattern for escape) substring(string from pattern)
44	substring_inner(string [from int] [for int])
45	tconvert(key text, value text)
46	to_single_byte(char)
47	translate(string text, from text, to text)
48	trim([leading |trailing |both] [characters] from string)
49	unistr(string)
50	vsize(expr)
51	PKG_UTIL.RAW_CAST_FROM_VARCHAR2
52	PKG_UTIL.LOB_CONVERTTOCLOB
53	PKG_UTIL.LOB_RAWTOTEXT
54	PKG_UTIL.LOB_TEXTTORAW
55	PKG_UTIL.RAW_CAST_TO_VARCHAR2
56	DBE_OUTPUT.PUT
57	DBE_OUTPUT.PUT_LINE

fix_func_selection

Parameter description: Specifies whether to optimize the function matching policy.

The catlist sequence issue occurs in this case: If a user-defined function conflicts with a system function, the function selected by the database depends on the registration sequence of the system function in the database system.

Parameter type: string.

Unit: none

Value range: **''** and **catlist**.

- '': No optimization is performed. The value is the same as that in versions earlier than 505.1.0.
- **catlist**: The catlist sequence is optimized. (The non-B-compatible mode is optimized. In non-B-compatible mode, system functions are always preferentially selected and executed.) The B-compatible mode is the same as that in versions earlier than 505.1.0. An error message indicating that the function is not unique may be displayed, or a system function may be selected for execution.

Default value:

- **catlist**: default value of the newly installed database
- '': default value of the database in versions earlier than 505.1.0 after the database is upgraded.

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

max_allowed_packet

Parameter description: Specifies the database configuration item in M-compatible mode.

Originally, this parameter specifies the maximum length of a data packet for communications between the database and the client in M-compatible mode. Currently, this parameter is only used to limit the maximum return values of some functions. It affects the upper limit of the return values of the string functions REPEAT, REPLACE, and SPACE, and affects the value of *N* in the CAST(expr AS BINARY(N)) and CONVERT(expr AS BINARY(N)) functions. This is a PGC_SIGHUP parameter.

Parameter type: integer.

Unit: byte

Value range: 1024 to 1073741824. The value must be a multiple of 1024. Otherwise, the value is rounded down to the nearest multiple.

Default value: **4194304**

Setting suggestion: Retain the default value **4194304**.

div_precision_increment

Parameter description: Specifies the database configuration item in M-compatible mode.

This is a session-level parameter, which is used to set the value of precision that the division result can improve. The final precision is the precision of the first operation parameter added by the value of this parameter.

Parameter type: integer.

Unit: none

Value range: 0 to 30

Default value: **4**

Setting suggestion: none.

enable_m_format_hook

Parameter description: Specifies the database configuration item in M-compatible mode.

This parameter specifies whether the hook in M-compatible mode takes effect.

Parameter type: Boolean.

Unit: none

Value range:

- **ON**: The M-compatible database allows the corresponding parser and executor hooks to be mounted.
- **OFF**: The M-compatible database does not mount the corresponding parser and executor hooks.

Default value: **ON**

Setting suggestion: This parameter is used only for external tools that are not fully adapted and cannot be used by users.

gs_format_behavior_compat_options

Parameter description: **gs_format_behavior_compat_options** is used to select configuration items of GaussDB internal system functions.

Parameter type: string.

Unit: none

Value range: Currently, only the compatibility configuration items listed in **Table 14-25** are supported. Compatibility configuration items are separated by commas (,).

Default value: **'sqrt_karatsuba'**

Table 14-25 gs_format_behavior compatibility configuration items

Configuration Item	Behavior
'sqrt_karatsuba'	• If this parameter is set, the Karatsuba square root algorithm is used when the sqrt function is called. The Karatsuba algorithm has higher performance but its precision may be different from that of the Newton iteration algorithm in rare cases. • If this parameter is not set, the default Newton iteration algorithm is used to calculate the square root for the sqrt algorithm.

Configuration Item	Behavior
'allow_textconcat_null'	• If this parameter is set, the value of the corresponding character string is returned when a character string is combined with the **null** value in PG-compatible mode. -- Run the following command in PG-compatible mode: gaussdb=# set gs_format_behavior_compat_options='allow_textconcat_null'; SET gaussdb=# select 'a' || null || 'b'; ?column? ---------- ab (1 row) • If this parameter is not set, **NULL** is returned when a character string is combined with the **null** value in PG-compatible mode.

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Determines whether some compatibility features are available. If you want to modify this parameter, make sure you understand its meaning and modify it with caution to avoid risks caused by misoperations.

14.3.16.3 Product Version of the Cloud Service

This section describes the version parameters of the cloud service.

product_version

Parameter description: Specifies the version of a cloud service product.

Parameter type: string

Unit: none

Value range: none

Default value: empty

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: The following rules must be met to avoid errors.

- The value can contain a maximum of 50 characters.
- The value cannot contain vertical bars (|), semicolons (;), ampersands (&), dollar signs ($), ampersands (>), ampersands (<), ampersands (`), backslashes (\), exclamation marks (!), or newline characters.

hotpatch_version

Parameter description: Specifies the hot patch version of a cloud service product.

Parameter type: string

Unit: none

Value range: none

Default value: empty

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: The following rules must be met to avoid errors.

- The value can contain a maximum of 1500 characters.
- The value cannot contain vertical bars (|), semicolons (;), ampersands (&), dollar signs ($), ampersands (>), ampersands (<), ampersands (`), backslashes (\), exclamation marks (!), or newline characters.

14.3.17 Fault Tolerance

This section describes parameters used for controlling how the server processes an error occurring in the database system.

exit_on_error

Parameter description: If this function is enabled, errors of the ERROR level will be escalated to PANIC errors, and core stacks will be generated. It is mainly used to locate problems and test services.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-2**.

Value range: Boolean

- **on**: Errors of the ERROR level will be escalated to PANIC errors.
- **off**: Errors of the ERROR level will not be escalated.

Default value: **off**

restart_after_crash

Parameter description: If this parameter is set to **on** and a backend process crashes, GaussDB automatically reinitializes the backend process.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-2**.

Value range: Boolean

- **on**: The availability of the database is maximized.

  In some circumstances (for example, when a management tool, such as xCAT, is used to manage GaussDB), setting this parameter to **on** maximizes the availability of the database.
- **off**: A management tool is enabled to obtain control permission and take proper measures when a backend process crashes.

Default value: **on**

omit_encoding_error

Parameter description: If this parameter is set to **on** and the client character set of the database is encoded in UTF-8 format, character encoding conversion errors will be recorded in logs. Additionally, converted characters that have conversion errors will be ignored and replaced with question marks (?).

This is a USERSET parameter. Set it based on instructions provided in **Table 14-2**.

Value range: Boolean

- **on**: Characters with conversion errors will be ignored and replaced with question marks (?), and error information will be recorded in logs.
- **off**: Characters with conversion errors cannot be converted and error information will be directly displayed.

Default value: **off**

max_query_retry_times

This parameter has been discarded in the current version.

cn_send_buffer_size

Parameter description: Specifies the size of the data buffer used for data transmission on the primary database node.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-2**.

Value range: an integer ranging from 8 to 128. The unit is KB.

Default value: **8KB**

data_sync_retry

Parameter description: Specifies whether to keep running the database when updated data fails to be written into disks by using the fsync function. In some OSs, no error is reported even if fsync fails after the second attempt. As a result, data is lost.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-2**.

Value range: Boolean

- **on**: The database keeps running and fsync is called again after fsync fails to synchronize data to the disk.
- **off**: A PANIC-level error is reported and the database is stopped after fsync fails.

Default value: **off**

remote_read_mode

Parameter description: Specifies whether to enable the remote read function. This function allows pages on the standby node to be read when reading pages on the primary node fails.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: enumerated values

- **off** indicates that the remote read function is disabled.
- **non_authentication** indicates that the remote read function is enabled but certificate authentication is not required.
- **authentication** indicates that the remote read function is enabled and certificate authentication is required.

Default value: **authentication**

14.3.18 Connection Pool Parameters

When a connection pool is used to access the database, database connections are established and then stored in the memory as objects during system running. When you need to access the database, no new connection is established. Instead, an existing idle connection is selected from the connection pool. After you finish accessing the database, the database does not disable the connection but puts it back into the connection pool. The connection can be used for the next access request.

cache_connection

Parameter description: Specifies whether to recycle the connections of a connection pool.

This parameter is a **SIGHUP** parameter. Set it based on instructions provided in **Table 14-2**.

Value range: Boolean

- **on** indicates that the connections of a connection pool will be recycled.
- **off** indicates that the connections of a connection pool will not be recycled.

Default value: **on**

14.3.19 Transaction

This section describes the settings and value ranges of database transaction parameters.

transaction_isolation

Parameter description: Specifies the isolation level of the current transaction.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string of case-sensitive characters. The values include:

- **serializable**: This value is equivalent to REPEATABLE READ in GaussDB.
- **read committed** indicates that only the data in committed transactions is read.
- **repeatable read** indicates that only the data committed before a transaction starts is read. Uncommitted data or data committed in other concurrent transactions cannot be read.
- **default**: The value is the same as that of **default_transaction_isolation**.

Default value: read committed

transaction_read_only

Parameter description: Specifies that the current transaction is a read-only transaction.

This parameter has a fixed value **on** during database restoration or on the standby node. Otherwise, set this parameter to the value of **default_transaction_read_only**.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the current transaction is a read-only transaction.
- **off** indicates that the current transaction can be a read/write transaction.

Default value: **off**

xc_maintenance_mode

Parameter description: Specifies whether the system is in maintenance mode.

This is a SUSET parameter. Set it based on method 3 provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the function is enabled.
- **off** indicates that the function is disabled.

NOTICE

Enable the maintenance mode with caution to avoid data inconsistencies in the database.

Default value: **off**

allow_concurrent_tuple_update

Parameter description: Specifies whether to allow concurrent update.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the function is enabled.
- **off** indicates that the function is disabled.

Default value: **on**

transaction_deferrable

Parameter description: Specifies whether to delay the execution of a read-only serial transaction without incurring an execution failure. Assume this parameter is set to **on**. When the server detects that the tuples read by a read-only transaction are being modified by other transactions, it delays the execution of the read-only transaction until the other transactions finish modifying the tuples. This parameter is reserved and does not take effect in this version. Similar to this parameter, the **default_transaction_deferrable** parameter is used to specify whether to allow delayed execution of a transaction.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: allowed.
- **off**: not allowed.

Default value: **off**

enable_show_any_tuples

Parameter description: This parameter is available only in a read-only transaction and is used for analysis. When this parameter is set to **on** or **true**, all versions of tuples in the table are displayed.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** or **true** indicates that all versions of tuples in the table are displayed.
- **off** or **false** indicates that no versions of tuples in the table are displayed.

Default value: **off**

NOTE

In the TOAST scenario, if DML operations (mainly INSERT+UPDATE or INSERT+DELETE) are performed before and after the REINDEX operation, and the read-only transaction is started and the GUC parameter is enabled after the REINDEX operation, historical data in the TOAST table or TOAST index table can be queried separately in the released version. However, when historical data in the TOAST column is queried in the main table, the error message "missing chunk number xxx" is displayed.

replication_type

Parameter description: Specifies whether the current database is deployed in standalone or one-primary-multiple-standby mode.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Do not set the value of this parameter.

Value range: 0 to 2

- **2** indicates the standalone mode. In this mode, no standby node can be added.
- **1** indicates the one-primary-multiple-standby mode, covering all scenarios. This mode is recommended.
- **0**: reserved parameter.

Default value: **1**

pgxc_node_name

Parameter description: Specifies the name of a node.

This is a POSTMASTER parameter. Set it based on **Table 14-2**.

When a standby node requests to replicate logs on the primary node, if the **application_name** parameter is not set, the **pgxc_node_name** parameter is used as the name of the streaming replication slot of the standby node on the primary node. The streaming replication slot is named in the following format: Value of this parameter_IP address of the standby node_Port number of the standby node. The IP address and port number of the standby node are obtained from the IP address and port number of the standby node specified by the **replconninfo** parameter. The maximum length of a streaming replication slot name is 61 characters. If the length of the concatenated string exceeds 61 characters, the truncated **pgxc_node_name** will be used for concatenation to ensure that the length of the streaming replication slot name is less than or equal to 61 characters.

CAUTION

After this parameter is modified, the database instance will fail to be connected. You are advised not to modify this parameter.

Value range: a string

Default value: current node name

enable_defer_calculate_snapshot

Parameter description: Specifies the delay in calculating **xmin** and **oldestxmin**. Calculation is triggered only when 1000 transactions are executed or the interval is 1s. If this parameter is set to **on**, the overhead of calculating snapshots can be reduced in heavy-load scenarios, but the progress of updating **oldestxmin** is slow, affecting tuple recycling. If this parameter is set to **off**, **xmin** and **oldestxmin** can be calculated in real time, but the overhead for calculating snapshots increases.

Parameter type: Boolean

Unit: none

Value range:

- **on** indicates that snapshots **xmin** and **oldestxmin** are calculated with a delay.

- **off** indicates that snapshots **xmin** and **oldestxmin** are calculated in real time.

Default value: **on**

Setting method: This is a SIGHUP parameter. Set it based on **Table 14-2**.

seqscan_csn_cache_num

Parameter description: Specifies the size of the CSN cache. The cache is used only when the Seq Scan scans the heap table page through the MVCC snapshot to determine the visibility. If this parameter is set to **0**, the CSN cache mechanism is not used.

This is a PGC_SIGHUP parameter. Set it based on **Table 14-2**.

Value range: an integer ranging from 0 to 1000.

Default value: **100**.

14.3.20 Replication Parameters of Two Database Instances

RepOriginId

Parameter description: This is a session-level GUC parameter. In bidirectional logical replication, set it to a non-zero value to avoid infinite data replication.

This is a USERSET parameter. Set it based on method 3 provided in **Table 14-1**.

Value range: an integer ranging from 0 to 2147483647

Default value: **0**

stream_cluster_run_mode

Parameter description: Specifies whether a DN is in the primary or standby instance in dual-instance streaming DR scenarios. For single-instance scenarios, the primary instance is used by default.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: enumerated values

- **cluster_primary** indicates that the node is in the primary instance.
- **cluster_standby** indicates that the node is in the standby instance.

Default value: **cluster_primary**

enable_roach_standby_cluster

Parameter description: Sets the standby database instances to read-only in dual-database instance mode. Only the sysadmin user can access this parameter.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the read-only mode is enabled for the standby database instances.
- **off** indicates that the read-only mode is disabled for the standby database instances. In this case, the standby database instances can be read and written.

Default value: **off**

hadr_process_type

Parameter description: Specifies a process ID of a streaming replication-based remote DR solution, an intra-city dual-DC HA solution, or an intra-city dual-DC HA solution that supports streaming replication-based remote DR.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-2**.

Value range: enumerated values.

- **none** indicates that there is no process.
- **failover** indicates that the DR database instance is being upgraded.
- **switchover_promote** indicates the process of upgrading the DR database instance to the primary one during the primary/standby database instance switchover according to a streaming replication-based remote DR solution.
- **switchover_demote** indicates the process of demoting the primary database instance to the DR one during the primary/standby database instance switchover according to the streaming replication-based remote DR solution.
- **dorado_failover** indicates the process of upgrading the DR database instance to the primary one according to the intra-city dual-DC HA solution.
- **dorado_switchover_demote** indicates the process of demoting the primary database instance to the DR one during the primary/standby database instance switchover according to the intra-city dual-DC HA solution.
- **dorado_failover_abnormal** indicates the process of promoting the DR database instance to the primary one when the shared disk of the primary database instance in the intra-city dual-DC HA solution is faulty.
- **dorado_failover_in_standby_stream** indicates the process of promoting the intra-city dual-DC standby database instance to the intra-city dual-DC primary database instance (also the remote DR database instance) when the remote database instance in the intra-city dual-DC HA solution that supports streaming replication-based remote DR is the remote DR primary database instance.
- **dorado_failover_abnormal_in_standby_stream** indicates the process of promoting the intra-city dual-DC standby database instance to the intra-city dual-DC primary database instance (also the remote DR database instance) when the remote database instance in the intra-city dual-DC HA solution that supports streaming replication-based remote DR is the remote DR primary database instance, and the shared disk of the intra-city dual-DC primary database instance (also the remote DR database instance) is faulty.

Default value: **none**

14.3.21 Developer Options

allow_system_table_mods

Parameter description: Specifies whether the structure of a system catalog or the name of a system schema can be modified.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the structure of the system catalog or the name of the system schema can be modified.
- **off** indicates that the structure of the system catalog or the name of the system schema cannot be modified.

Default value: **off**

CAUTION

You are advised not to change the default value of this parameter. If this parameter is set to **on**, system catalogs may be damaged and the database may fail to be started.

allow_create_sysobject

Parameter description: Specifies whether objects such as functions, stored procedures, synonyms, aggregate functions, and operators can be created or modified in the system schema. The system schema refers to the schema provided by the database after initialization, excluding the public schema. The OID of the system schema is usually smaller than 16384.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that initial users and system administrators can create or modify objects such as functions, stored procedures, synonyms, and aggregate functions in the system schema. The sysadmin user has permissions to create or replace, alter, grant, and revoke system objects by default. For details about whether other users are allowed to create these objects, see the permission requirements of the corresponding schema.
- **off** indicates that all users are not allowed to create or modify objects such as functions, stored procedures, synonyms, aggregate functions, and operators in the system schema. The sysadmin user does not have permissions to create or replace, alter, grant, and revoke system objects by default.

Default value: **on**

debug_assertions

Parameter description: Specifies whether to enable various assertion checks. It can assist in debugging. When an exception or a crash occurs, enable this

parameter to identify programming defects. To use this parameter, the macro USE_ASSERT_CHECKING must be defined (through the configure option **--enable-cassert**) during the GaussDB compilation.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that various assertion checks are enabled.
- **off** indicates that various assertion checks are disabled.

NOTE

If you compile GaussDB with the assertion check enabled, **debug_assertions** is set to **on** by default.

Default value: **off**

ignore_checksum_failure

Parameter description: Specifies whether to ignore check failures (but still generates an alarm) and continues reading data. Continuing reading data may result in breakdown, damaged data being transferred or hidden, failure of data recovery from remote nodes, or other serious problems. You are advised not to modify the settings.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that data check errors are ignored.
- **off** indicates that data check errors are reported.

Default value: **off**

ignore_system_indexes

Parameter description: Specifies whether to ignore system indexes when reading system catalog (but still update the indexes when modifying the tables).

This is a BACKEND parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

This parameter is useful for recovering data from tables whose system indexes are damaged.

Value range: Boolean

- **on** indicates that system indexes are ignored.
- **off** indicates that system indexes are not ignored.

Default value: **off**

post_auth_delay

Parameter description: Specifies the delay in the connection to the server after a successful authentication. Developers can attach a debugger to the server startup process.

This is a BACKEND parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 2147. The unit is s.

Default value: **0**

NOTE

This parameter is used only for commissioning and fault locating. To prevent impact on service running, ensure that the default value **0** is used in the production environment. If this parameter is set to a value other than **0**, the cluster may be abnormal due to a long authentication delay.

pre_auth_delay

Parameter description: Specifies the period of delaying authentication after the connection to the server is started. Developers can attach a debugger to the authentication procedure.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 60. The unit is s.

Default value: **0**

NOTE

This parameter is used only for commissioning and fault locating. To prevent impact on service running, ensure that the default value **0** is used in the production environment. If this parameter is set to a value other than **0**, the cluster may be abnormal due to a long authentication delay.

trace_notify

Parameter description: Specifies whether to enable the function of generating debugging output for the **LISTEN** and **NOTIFY** commands. The level of **client_min_messages** or **log_min_messages** must be **debug1** or lower so that debugging output can be recorded in the client or server logs, respectively.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the output function is enabled.
- **off** indicates that the output function is disabled.

Default value: **off**

trace_recovery_messages

Parameter description: Specifies whether to enable logging of recovery-related debugging output. This parameter allows users to overwrite the normal setting of

log_min_messages, but only for specific messages. This is intended for the use in debugging the standby node.

This is a SIGHUP parameter. Set it based on instructions provided in Table 14-1.

Value range: enumerated values. Valid values include **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, and **log**. For details about the parameter values, see log_min_messages.

Default value: **log**

NOTE

- **log** indicates that recovery-related debugging information will not be logged.
- Except the default value **log**, each of the other values indicates that recovery-related debugging information at the specified level will also be logged. Common settings of **log_min_messages** enable logs to be unconditionally recorded into server logs.

trace_sort

Parameter description: Specifies whether to print information about resource usage during sorting operations. This parameter is available only when the macro TRACE_SORT is defined during the GaussDB compilation. However, TRACE_SORT is currently defined by default.

This is a USERSET parameter. Set it based on instructions provided in Table 14-1.

Value range: Boolean

- **on** indicates that the function is enabled.
- **off** indicates that the function is disabled.

Default value: **off**

zero_damaged_pages

Parameter description: Specifies whether to detect a damaged page header that causes GaussDB to report an error, aborting the current transaction.

This is a SUSET parameter. Set it based on instructions provided in Table 14-1.

Value range: Boolean

Setting this parameter to **on** causes the system to report a warning, zero out the damaged page, and continue processing. This behavior will destroy data, including all the rows on the damaged page. However, it allows you to bypass the error and retrieve rows from any undamaged pages that may be present in the table. Therefore, it is useful for restoring data if corruption has occurred due to a hardware or software error. In most cases, you are advised not to set this parameter to **on** if you want to restore data from damaged pages.

Default value: **off**

remotetype

Parameter description: Specifies the remote connection type.

This is a BACKEND parameter. Set it based on instructions provided in Table 14-1.

Value range: enumerated values. Valid values are **application**, **datanode**, and **internaltool**.

Default value: **application**

max_user_defined_exception

Parameter description: Specifies the maximum number of exceptions. The default value cannot be changed.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer. Currently, only the fixed value **1000** is supported.

Default value: **1000**

enable_fast_numeric

Parameter description: Specifies whether to enable optimization for numeric data calculation. Calculation of numeric data is time-consuming. Numeric data is converted into int64- or int128-type data to improve numeric data calculation performance.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** or **true** indicates that optimization for numeric data calculation is enabled.
- **off** or **false** indicates that optimization for numeric data calculation is disabled.

Default value: **on**

enable_compress_spill

Parameter description: Specifies whether to enable the compression function of writing data to disk.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** or **true** indicates that optimization for writing data to disk is enabled.
- **off** or **false** indicates that optimization for writing data to a disk is disabled.

Default value: **on**

resource_track_log

Parameter description: Specifies the log level of self-diagnosis. Currently, this parameter takes effect only on multi-column statistics.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string

- **summary**: Brief diagnosis information is displayed.

- **detail**: Detailed diagnosis information is displayed.

Currently, the two parameter values differ only when there is an alarm about multi-column statistics not collected. If the parameter is set to **summary**, such an alarm will not be displayed. If it is set to **detail**, such an alarm will be displayed.

Default value: **summary**

show_acce_estimate_detail

Parameter description: The estimation information is generally used by O&M personnel during maintenance, and it may affect the output display of the EXPLAIN statement. Therefore, this parameter is disabled by default. The estimation information is displayed only if the **verbose** option of the EXPLAIN statement is enabled. (Due to specification changes, the current version no longer supports this feature. Do not use it.)

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the estimation information is displayed in the output of the EXPLAIN statement.
- **off** indicates that the estimation information is not displayed in the output of the EXPLAIN statement.

Default value: **off**

NOTE

The current version does not support database acceleration. Therefore, this parameter does not take effect after being set.

support_batch_bind

Parameter description: Specifies whether to batch bind and execute PBE statements through interfaces such as JDBC, ODBC, and libpq.

Parameter type: Boolean

Unit: none

Value range:

- **on** indicates that batch binding and execution are used.
- **off** indicates that batch binding and execution are not used.

Default value: **on**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

numa_distribute_mode

Parameter description: Specifies the distribution of some shared data and threads among NUMA nodes. This parameter is used to optimize the performance of

large-scale Arm servers with multiple NUMA nodes. Generally, you do not need to set this parameter.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. The valid values are **'none'** and **'all'**.

- **none** indicates that this function is disabled.
- **all** indicates that some shared data and threads are distributed to different NUMA nodes to reduce the number of remote access times and improve performance. Currently, this function applies only to Arm servers with multiple NUMA nodes. All NUMA nodes must be available for database processes. You cannot select only some NUMA nodes.

NOTE

In the current version, **numa_distribute_mode** cannot be set to **all** on the x86 architecture.

Default value: **'none'**

log_pagewriter

Parameter description: Specifies whether to display the page refresh information of a thread and details about an incremental check point after the incremental check point is enabled. You are advised not to set this parameter to **true** because a large amount of information will be generated.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

Default value: **on**

advance_xlog_file_num

Parameter description: Specifies the number of Xlog files that are periodically initialized in advance in the background. This parameter is used to prevent the Xlog file initialization from affecting the performance during transaction commit. However, such a fault may occur only when the system is overloaded. Therefore, you do not need to set this parameter.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 1000000. The value **0** indicates that initialization is not performed in advance. For example, the value **10** indicates that the background thread periodically initializes 10 Xlog files in advance based on the write location of the current Xlog.

Default value: **0**

enable_beta_opfusion

Parameter description: Specifies whether to accelerate the execution of SQL statements, such as aggregate functions and sorting in TPC-C when **enable_opfusion** is set to **on**.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **off**

enable_csqual_pushdown

Parameter description: Specifies whether to push down filter criteria for a rough check during query.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that a rough check is performed with filter criteria delivered during query.
- **off** indicates that a rough check is performed without filter criteria delivered during query.

Default value: **on**

string_hash_compatible

Parameter description: Specifies whether to use the same method to calculate char-type hash values and varchar- or text-type hash values. Based on the setting of this parameter, you can determine whether a redistribution is required when a distribution key is converted from a char-type data distribution into a varchar- or text-type data distribution.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the same calculation method is used and a redistribution is not required.
- **off** indicates that different calculation methods are used and a redistribution is required.

NOTE

Calculation methods differ in the length of input strings used for calculating hash values. (For a char-type hash value, spaces following a string are not counted as the length. For a text- or varchar-type hash value, the spaces are counted.) The hash value affects the calculation result of queries. To avoid query errors, do not modify this parameter during database running once it is set.

Default value: **off**

pldebugger_timeout

Parameter description: Specifies the timeout interval for the pldebugger server to wait for a response from the debug end.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to 86400. The unit is second.

Default value: **15min**

plsql_show_all_error

Parameter description: Specifies whether to skip errors and continue compiling PL/SQL objects. For details about the impact, see "Schema > DBE_PLDEVELOPER" in *Developer Guide*.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

Default value: **off**

ustore_attr

Parameter description: This parameter is used to control the information statistics of Ustore tables, rollback type, UB-tree index type, and data verification during the running of key modules (including data, indexes, rollback segments, and replay). This parameter helps R&D engineers locate faults.

Parameter type: string

Unit: none

Value range: This parameter is set in key-value mode. The mapping between keys and values is as follows: If multiple key-value pairs are used, use semicolons (;) to separate them. For example, ustore_attr='ustore_verify_level=FAST;ustore_verify_module=UPAGE:UBTREE:UNDO:REDO'.

- **ustore_verify_level**: verification level.

  Value range: a string of case-insensitive characters. For details, see the following table.

  Table 14-26 Parameter value meaning of ustore_verify_level

Parameter Value	Description
NONE	**NONE** indicates that the verification function is disabled. You are advised to enable this function to test performance.
FAST	Indicates fast verification. Few content is to be verified and the impact on performance is minimized. The value is **NORMAL** for compatibility with earlier versions.
COMPLETE	Indicates complete verification. The verification content is the largest and the performance is greatly affected. The value is **SLOW** for compatibility with earlier versions.

Default value: **FAST**

- **ustore_verify_module**: module that controls verification.

  Value range: The value can be one or more of **UPAGE**, **UBTREE**, **UNDO**, **REDO**, and **ROACH**, or it can be **ALL** or **NULL** (case-insensitive). When multiple values are used, separate them with colons (:). For example, **ustore_verify_module=UPAGE:UBTREE:UNDO:REDO**.

  When the ROACH module is enabled, the value of the **ustore_verify_level** parameter is ignored during the ROACH backup. By default, the level of verification is the highest and the performance is greatly affected. Therefore, exercise caution when using this parameter.

  Table 14-27 Parameter value meaning of ustore_verify_module

Parameter Value	Description
UPAGE	Indicates that data page verification is enabled.
UBTREE	Indicates that UB-tree index verification is enabled.
UNDO	Indicates that rollback segment data verification is enabled.
REDO	Indicates that data page verification for the REDO process is enabled.
ROACH	This parameter is discarded. The verification has been removed from the underlying logic. **ustore_verify_module** can be set to **roach** but it does not take effect.
ALL	Indicates that data verification is enabled for the UPAGE, UBTREE, UNDO, REDO, and ROACH modules.
NULL	Indicates that data verification for the UPAGE, UBTREE, UNDO, REDO, and ROACH modules is disabled.

  Default value: **UPAGE:UBTREE:UNDO**

- **index_trace_level**: specifies whether to enable index tracing and controls the printing level. After this function is enabled, information about index tuples that meet the conditions is printed based on the printing level during index scan.

  Value range: a string. The values are described in the following table.

  Default value: **NO**

  Table 14-28 Parameter value meaning of index_trace_level

Parameter Value	Description
NO	No additional information is printed.

Parameter Value	Description
NORMAL	Information about visible index tuples is printed, including: • ID and offset of the index page where the current index tuple is located • Current tuple status • TID and partOid corresponding to the current tuple • xmin and xmax information corresponding to the current tuple • Current tuple content (if **enable_log_tuple** is set to **on**).
VISIBILITY	On the basis of **NORMAL**, the information about the index tuples that do not pass the visibility check is printed and whether the index tuples are visible is marked.
SHOWHIKEY	On the basis of **VISIBILITY**, the system tries to print the information about the HIKEY tuple on the page.
ALL	Information about all tuples on the scanned index page is printed.

- **enable_log_tuple**: specifies whether to print the contents of related tuples when printing log-level prompts for troubleshooting and locating.

  Value range: **on** or **off** (case-insensitive)

  Default value: **off**

  Note: This parameter has been discarded.
- **enable_ustore_sync_rollback**: specifies whether to enable synchronous rollback for Ustore tables.

  Value range: Boolean

  Default value: **true**
- **enable_ustore_async_rollback**: specifies whether to enable asynchronous rollback for Ustore tables.

  Value range: Boolean

  Default value: **true**
- **enable_ustore_page_rollback**: specifies whether to enable page rollback for Ustore tables.

  Value range: Boolean

  Default value: **true**
- **enable_ustore_partial_seqscan**: specifies whether to enable partial scan for Ustore tables.

  Value range: Boolean

  Default value: **false**
- **enable_candidate_buf_usage_count**: specifies whether to enable buffer usage statistics.

Value range: Boolean

Default value: **false**

- **ustats_tracker_naptime**: specifies the interval for collecting statistics on Ustore tables.

  Value range: [1,INT_MAX/1000]

  Default value: **20**, in seconds.

- **umax_search_length_for_prune**: specifies the maximum search depth of the prune operation on the Ustore table.

  Value range: [1,INT_MAX/1000]

  Default value: **10**

NOTE

When setting **ustore_attr**, do not leave spaces or other characters before and after the equal sign (=) between key and value, for example, **ustore_attr='ustore_verify_level = FAST;**. Otherwise, the parameter is invalid during kernel code verification and the parameter setting fails.

Default value: an empty string

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

CAUTION

Exercise caution when setting the **ustore_attr** parameter. You are advised to modify this parameter with the assistance of engineers.

index_txntype

Parameter description: Determines whether the index type of UB-tree is PCR or RCR. PCR supports flashback query based on indexes, but RCR does not.

Parameter type: string

Unit: none

Value range: **'PCR'** or **'RCR'**.

Default value: an empty string

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: You are advised to set it to "'RCR".

enable_segment_remain_cleanup

Parameter description: Specifies which the residual segment-page cleanup feature is to be enabled.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: indicates that the old type of residual segment-page cleanup is enabled.
  - Query interfaces: gs_stat_remain_segment_info and gs_local_stat_remain_segment_info.
  - Cleanup interfaces: gs_free_remain_segment and gs_local_free_remain_segment.
- **off**: indicates that the new type of residual segment-page cleanup is enabled.
  - Query interfaces: GS_SEG_SPC_REMAIN_SEGMENTS and GS_SEG_SPC_REMAIN_EXTENTS.
  - Cleanup interfaces: gs_seg_free_spc_remain_segment and gs_seg_free_spc_remain_extent.

CAUTION

Do not change the cleanup mode unless necessary. Otherwise, residual segment-page data may fail to be cleaned up or other exceptions may occur.

Default value: **off**

convert_illegal_char_mode

Parameter description: Specifies the placeholders of invalid characters that can be displayed on the client.

Parameter type: string

Unit: none

Value range: 95 characters whose decimal codes range from 32 to 126 in the ASCII coding table.

Default value: **'?'**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none

default_segment

Parameter description: Specifies whether to create a segment-page table by default.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: A segment-page table is created by default when the segment field is not specified.
- **off**: A page table is created by default when the segment field is not specified.

Default value: **off**

Setting method: This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none.

CAUTION

- If **--undostoragetype** is set to **'page'** or is not set during database initialization, the data of Ustore is forcibly stored in page mode, but Astore is not affected.

14.3.22 Auditing

14.3.22.1 Audit Switch

audit_enabled

Parameter description: Specifies whether to enable or disable the audit thread. After the audit thread is enabled, the auditing information written by the background thread can be read from the pipe and written into audit files.

Parameter type: Boolean

Unit: none

Value range:

- **on** indicates that the auditing function is enabled.
- **off** indicates that the auditing function is disabled.

Default value: **on**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

audit_directory

Parameter description: Specifies the storage directory of audit files. The path can be relative to the **data** directory or an absolute path. Only the sysadmin user can access this parameter.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string

Default value: pg_audit. If **om** is used for database deployment, audit logs are stored in *$GAUSSLOG*/**pg_audit/***Instance name*.

NOTICE

- You need to set different audit file directories for different DNs. Otherwise, audit logs will be abnormal.
- If the value of **audit_directory** in the configuration file is an invalid path, the audit function cannot be used.

NOTE

- Valid path: Users have read and write permissions on the path.
- Invalid path: Users do not have read or write permission on the path.

audit_data_format

Parameter description: Audits the format of log files. Currently, only the binary format is supported. Only the sysadmin user can access this parameter.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string

Default value: **binary**

audit_rotation_interval

Parameter description: Specifies the interval of creating an audit log file. If the difference between the current time and the time when the previous audit log file is created is greater than the value of this parameter, a new audit log file will be generated.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to *INT_MAX*/60. The unit is min.

Default value: 1d

NOTICE

Do not adjust this parameter unless necessary. Otherwise, **audit_resource_policy** may fail to take effect. To control the storage space and time of audit logs, set the **audit_resource_policy**, **audit_space_limit**, and **audit_file_remain_time** parameters.

audit_rotation_size

Parameter description: Specifies the maximum capacity of an audit log file. If the total number of messages in an audit log exceeds the value of this parameter, the server will generate a new audit log file.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1024 to 1048576. The unit is KB.

Default value: **10 MB**

NOTICE

- Do not adjust this parameter unless necessary. Otherwise, **audit_resource_policy** may fail to take effect. To control the storage space and time of audit logs, set the **audit_resource_policy**, **audit_space_limit**, and **audit_file_remain_time** parameters.
- If the space occupied by a single record in an audit log file exceeds the value of this parameter, the log file is regarded as an invalid log file.

audit_resource_policy

Parameter description: Specifies the policy for determining whether audit logs are preferentially stored by space or time.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that audit logs are preferentially stored by space. A maximum of **audit_space_limit** logs can be stored.
- **off** indicates that audit logs are preferentially stored by time. A minimum duration of **audit_file_remain_time** logs must be stored.

Default value: **on**

audit_file_remain_time

Parameter description: Specifies the minimum duration required for recording audit logs. This parameter is valid only when **audit_resource_policy** is set to **off**.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 730. The unit is day. **0** indicates that the storage duration is not limited.

Default value: **90**

audit_space_limit

Parameter description: Specifies the total disk space occupied by audit files.

Parameter type: integer

Unit: KB

Value range: 1024 KB to 1024 GB

Default value: **1GB**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

In the multi-audit thread scenario, the minimum disk space occupied by audit files is the product of values of **audit_thread_num** and **audit_rotation_size**. Ensure that the value of **audit_space_limit** is greater than the product of values of **audit_thread_num** and **audit_rotation_size**.

audit_file_remain_threshold

Parameter description: Specifies the maximum number of audit files in the audit directory.

Parameter type: integer

Unit: none

Value range: an integer ranging from 100 to 1048576

Default value:

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. For details, see the following notes:

1048576 (196-core CPU/1536 GB memory, 128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 64-core CPU/512 GB memory, 60-core CPU/480 GB memory, 32-core CPU/256 GB memory, 16-core CPU/128 GB memory, 8-core CPU/64 GB memory, 4-core CPU/32 GB memory); **1024** (4-core CPU/16 GB memory)

NOTICE

- Ensure that this parameter is set to **1048576**. Do not adjust this parameter unless necessary. Otherwise, **audit_resource_policy** may fail to take effect. To control the storage space and time of audit logs, set the **audit_resource_policy**, **audit_space_limit**, and **audit_file_remain_time** parameters.
- In the multi-audit thread scenario, do not adjust this parameter unless necessary. Ensure that the value of this parameter is greater than or equal to the value of **audit_thread_num**. Otherwise, the audit function cannot be used and the database is abnormal.

audit_thread_num

Parameter description: Specifies the number of threads used for auditing.

Parameter type: integer

Unit: none

Value range: 1 to 48

Default value: **1**

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

- The larger the number of threads, the greater the impact on system performance.
- The larger the number of threads, the more CPU and I/O resources are occupied.

NOTICE

- When **audit_dml_state** is enabled and high performance is required, you are advised to increase the value of this parameter to ensure that audit messages can be processed and recorded in a timely manner.

14.3.22.2 User and Permission Audit

audit_login_logout

Parameter description: Specifies whether to audit users' login (including login success and failure) and logout.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 7.

- **0** indicates that the function of auditing users' logins and logouts is disabled.
- **1** indicates that only successful user logins are audited.
- **2** indicates that only failed user logins are audited.
- **3** indicates that successful and failed user logins are audited.
- **4** indicates that only user logouts are audited.
- **5** indicates that successful user logouts and logins are audited.
- **6** indicates that failed user logouts and logins are audited.
- **7** indicates that successful user logins, failed user logins, and logouts are audited.

Default value: **7**

audit_database_process

Parameter description: Specifies whether to audit the database startup, stop, switchover, and recovery.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer, **0** or **1**.

- **0** indicates that the function of auditing database startup, stop, switchover, and recovery is disabled.
- **1** indicates that the function of auditing database startup, stop, switchover, and recovery is enabled.

Default value: **1**

NOTICE

When the database is started, the standby DN is promoted to primary. Therefore, the DN type in the audit log is **system_switch** when the DN is started.

audit_user_locked

Parameter description: Specifies whether to audit the users' locking and unlocking.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer, **0** or **1**.

- **0** indicates that the function of auditing user's locking and unlocking is disabled.
- **1** indicates that the function of auditing user's locking and unlocking is enabled.

Default value: **1**

audit_user_violation

Parameter description: Specifies whether to audit the access violation operations of a user.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer, **0** or **1**.

- **0** indicates that the function of auditing the access violation operations of a user is disabled.
- **1** indicates that the function of auditing the access violation operations of a user is enabled.

Default value: **0**

audit_grant_revoke

Parameter description: Specifies whether to audit the granting and revoking of user permissions.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer, **0** or **1**.

- **0** indicates that the function of auditing the granting and recycling of a user's permission is disabled.
- **1** indicates that the function of auditing the granting and recycling of a user's permission is enabled.

Default value: **1**

audit_security_label

Parameter description: Specifies whether to audit the creation, deletion, and application of user security labels.

Parameter type: integer.

Unit: none

Value range: **0** or **1**

- **0** indicates that the creation, deletion, and application of user security labels are not audited.
- **1** indicates that the creation, deletion, and application of user security labels are audited.

Default value: **0**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. If this parameter is enabled, the system performance may be affected.

audit_internal_event

Parameter description: Specifies whether to audit the connections and operations of internal tools CM Agent, gs_clean, and WDRXdb.

Parameter type: Boolean.

Unit: none

Value range:

- **off**: The login, logout, and operations of internal tools CM Agent, gs_clean, and WDRXdb are not audited.
- **on**: The login, logout, and operations of internal tools CM Agent, gs_clean, and WDRXdb are audited.

Default value: **off**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

NOTICE

Internal tools on the database server also generate audit logs. By default, **audit_internal_event** is disabled to reduce space occupied by audit logs and improve audit log query performance.

full_audit_users

Parameter description: Specifies the full audit user list. Audit logs are recorded for all auditable operations performed by users in the list.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. Use commas (,) to separate multiple usernames.

Default value: an empty string.

no_audit_client

Parameter description: Specifies the names and IP addresses of clients that do not need to be audited. The parameter format is *client name*@*IP address*, which is the same as that of the **client_conninfo** column in the pg_query_audit function, for example, **cm_agent@127.0.0.1 or gs_clean@127.0.0.1**.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. Use commas (,) to separate multiple configuration items.

Default value: an empty string.

NOTICE

- If the executed SQL statement meets the configuration requirements of **full_audit_users** and **no_audit_client**, the **no_audit_client** is preferentially configured and no audit logs are recorded.
- Audit logs are generated for communication among tools or nodes in the database server. To save space occupied by audit logs and improve the query performance of audit logs, you can configure the **no_audit_client** parameter not to audit the low-risk scenarios.

14.3.22.3 Operation Auditing

audit_system_object

Parameter description: Specifies whether to audit the operations such as CREATE, DROP, and ALTER on database objects. Database objects include databases, users, schemas, and tables. You can change the value of this parameter to audit only the operations on required database objects. In the scenario where the leader node is forcibly elected, you are advised to set **audit_system_object** to the maximum value and audit all DDL objects.

Parameter type: integer

Unit: none

Value range: 0 to 536870911

- **0** indicates that the operations such as CREATE, DROP, and ALTER are not audited.
- A non-zero value indicates that the operations such as CREATE, DROP, and ALTER on a certain or some database objects are audited.

Value description:

The value of this parameter is calculated by 29 binary bits. The 29 binary bits represent 29 types of database objects. If the corresponding binary bit is set to **0**,

the operations such as CREATE, DROP, and ALTER on corresponding database objects are not audited. If it is set to **1**, the operations such as CREATE, DROP, and ALTER are audited. For details about the audit contents represented by these 29 binary bits, see **Table 14-29**.

When SQL patches are audited and **audit_dml_state_select** is enabled, an SQL patch operation will be audited twice and recorded as DML and DDL operations in the audit log, respectively.

Default value: **67121159** (decimal), corresponding to 00100 0000 0000 0011 0000 0000 0111 in binary, indicating that DDL operations on the four database objects DATABASE, SCHEMA, USER, and SQLPatch are audited.

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Set the type of database objects to be audited based on service requirements.

- The more types of objects to be audited, the greater the impact on system performance.
- The more objects to be audited, the more CPU and I/O resources are occupied.

Table 14-29 Meaning of each value for the **audit_system_object** parameter

Binary Bit	Description	Value Range
Bit 0	Specifies whether to audit the CREATE, DROP, and ALTER operations on databases.	• **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 1	Specifies whether to audit the CREATE, DROP, and ALTER operations on schemas.	• **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 2	Specifies whether to audit the CREATE, DROP, and ALTER operations on users.	• **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.

Binary Bit	Description	Value Range
Bit 3	Specifies whether to audit the CREATE, DROP, ALTER, and TRUNCATE operations on tables.	• **0** indicates that the CREATE, DROP, ALTER, and TRUNCATE operations on these objects are not audited. • **1** indicates that the CREATE, DROP, ALTER, and TRUNCATE operations on these objects are audited.
Bit 4	Specifies whether to audit the CREATE, DROP, and ALTER operations on indexes.	• **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 5	Specifies whether to audit the CREATE and DROP operations on VIEW and MATVIEW objects.	• **0** indicates that the CREATE and DROP operations on these objects are not audited. • **1** indicates that the CREATE and DROP operations on these objects are audited.
Bit 6	Specifies whether to audit the CREATE, DROP, and ALTER operations on triggers.	• **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 7	Specifies whether to audit the CREATE, DROP, and ALTER operations on procedures/functions.	• **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 8	Specifies whether to audit the CREATE, DROP, and ALTER operations on tablespaces.	• **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 9	Specifies whether to audit the CREATE, DROP, and ALTER operations on resource pools.	• **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.

Binary Bit	Description	Value Range
Bit 10	Specifies whether to audit the CREATE, DROP, and ALTER operations on workloads.	• **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 11	Specifies whether to audit the CREATE, DROP, and ALTER operations on server objects.	• **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 13	Reserved	-
Bit 14	Specifies whether to audit the CREATE, DROP, and ALTER operations on ROW LEVEL SECURITY objects.	• **0** indicates that the CREATE, DROP, and ALTER operations on these objects are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on these objects are audited.
Bit 15	Specifies whether to audit the CREATE, DROP, and ALTER operations on types.	• **0** indicates that the CREATE, DROP, and ALTER operations on types are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on types are audited.
Bit 16	Specifies whether to audit the CREATE, DROP, and ALTER operations on text search objects (CONFIGURATION and DICTIONARY).	• **0** indicates that the CREATE, DROP, and ALTER operations on text search objects are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on text search objects are audited.
Bit 17	Specifies whether to audit the CREATE, DROP, and ALTER operations on directories.	• **0** indicates that the CREATE, DROP, and ALTER operations on directories are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on directories are audited.

Binary Bit	Description	Value Range
Bit 18	Specifies whether to audit the CREATE, DROP, and ALTER operations on synonyms.	• **0** indicates that the CREATE, DROP, and ALTER operations on synonyms are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on synonyms are audited.
Bit 19	Specifies whether to audit the CREATE, DROP, and ALTER operations on sequences.	• **0** indicates that the CREATE, DROP, and ALTER operations on sequences are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on sequences are audited.
Bit 20	Specifies whether to audit the CREATE and DROP operations on CMK and CEK objects.	• **0** indicates that the CREATE, ALTER, and DROP operations on CMKs and CEKs are not audited. • **1** indicates that the CREATE, ALTER, and DROP operations on CMKs and CEKs are audited.
Bit 21	Specifies whether to audit the CREATE, DROP, and ALTER operations on PACKAGE objects.	• **0** indicates that the CREATE, DROP, and ALTER operations on packages are not audited. • **1** indicates that the operations are audited.
Bit 22	Specifies whether to audit the CREATE and DROP operations on MODEL objects.	• **0** indicates that the CREATE and ALTER operations on models are not audited. • **1** indicates that the CREATE and DROP operations are audited.
Bit 24	Specifies whether to audit the ALTER and DROP operations on the gs_global_config objects.	• **0** indicates that the ALTER and DROP operations on the gs_global_config objects are not audited. • **1** indicates that the ALTER and DROP operations on the gs_global_config objects are audited.
Bit 25	Specifies whether to audit the CREATE, DROP, and ALTER operations on FOREIGN DATA WRAPPER objects.	• **0** indicates that the CREATE, DROP, and ALTER operations on FOREIGN DATA WRAPPER objects are not audited. • **1** indicates that the CREATE, DROP, and ALTER operations on FOREIGN DATA WRAPPER objects are audited.

Binary Bit	Description	Value Range
Bit 26	Specifies whether to audit the CREATE, ENABLE, DISABLE, and DROP operations on SQL patches.	• **0** indicates that the CREATE, ENABLE, DISABLE, and DROP operations on SQL patches are not audited. • **1** indicates that the CREATE, ENABLE, DISABLE, and DROP operations on SQL patches are audited.
Bit 27	Specifies whether to audit the CREATE, ALTER, and DROP operations on events.	• **0** indicates that the CREATE, ENABLE, DISABLE, and DROP operations on events are not audited. • **1** indicates that the CREATE, ENABLE, DISABLE, and DROP operations on events are not audited.
Bit 28	Specifies whether to audit the CREATE, ALTER, and DROP operations on database links. Currently, the database link function is not supported.	• **0** indicates that the CREATE, ALTER, and DROP operations on database links are not audited. • **1** indicates that the CREATE, ALTER, and DROP operations on database links are audited.

audit_dml_state

Parameter description: Specifies whether to audit the INSERT, UPDATE, DELETE, and MERGE operations on a specific table.

Parameter type: integer

Unit: none

Value range: **0** or **1**

- **0** indicates that the INSERT, UPDATE, DELETE, and MERGE operations on a specific table are not audited.
- **1** indicates that the INSERT, UPDATE, DELETE, and MERGE operations on a specific table are audited.

Default value: **0**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

audit_dml_state_select

Parameter description: Specifies whether to audit the SELECT operation.

Parameter type: integer

Unit: none

Value range: **0** or **1**

- **0** indicates that the SELECT auditing function is disabled.
- **1** indicates that the SELECT auditing function is enabled.

Default value: **0**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

audit_function_exec

Parameter description: Specifies whether to record the audit information during the execution of the stored procedures, anonymous blocks, or user-defined functions (excluding system functions).

Parameter type: integer

Unit: none

Value range: **0** or **1**

- **0** indicates that the stored procedures, anonymous blocks, or user-defined functions (excluding system functions) are not audited.
- **1** indicates that the stored procedures, anonymous blocks, or user-defined functions (excluding system functions) are audited.

Default value: **0**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

audit_system_function_exec

Parameter description: Specifies whether to record audit logs when system functions in the whitelist are executed.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer, **0** or **1**

- **0** indicates that the function of auditing the execution of system functions is disabled.
- **1** indicates that the function of auditing system function execution is enabled.

Default value: **0**

The following table lists the whitelist of system functions that can be recorded and audited.

set_working_grand_version_num_manually	set_config	pg_cancel_backend	pg_cancel_session	pg_reload_conf	pg_rotate_logfile
pg_terminate_session	pg_terminate_backend	pg_create_restore_point	pg_start_backup	pg_stop_backup	pg_switch_xlog
pg_cbm_rotate_file	pg_cbm_get_merged_file	pg_cbm_recycle_file	pg_enable_delay_ddl_recycle	pg_disable_delay_ddl_recycle	gs_roach_stop_backup
gs_roach_enable_delay_ddl_recycle	gs_roach_disable_delay_ddl_recycle	gs_roach_switch_xlog	pg_last_xlog_receive_location	pg_xlog_replay_pause	pg_xlog_replay_resume
gs_pitr_clean_history_global_barriers	gs_pitr_archive_slot_force_advance	pg_create_physical_replication_slot_extern	gs_set_obs_delete_location	gs_hadr_do_switchover	gs_set_obs_delete_location_with_slotname
gs_streaming_dr_in_switchover	gs_upload_obs_file	gs_download_obs_file	gs_set_obs_file_context	gs_get_hadr_key_cn	pg_advisory_lock
pg_advisory_lock_shared	pg_advisory_unlock	pg_advisory_unlock_shared	pg_advisory_unlock_all	pg_advisory_xact_lock	pg_advisory_xact_lock_shared
pg_try_advisory_lock	pg_try_advisory_lock_shared	pg_try_advisory_xact_lock	pg_try_advisory_xact_lock_shared	pg_create_logical_replication_slot	pg_drop_replication_slot
pg_logical_slot_peek_changes	pg_logical_slot_get_changes	pg_logical_slot_get_binary_changes	pg_replication_slot_advance	pg_replication_origin_create	pg_replication_origin_drop
pg_replication_origin_session_setup	pg_replication_origin_session_reset	pg_replication_origin_session_progress	pg_replication_origin_xact_setup	pg_replication_origin_xact_reset	pg_replication_origin_advance
local_space_shrink	gs_space_shrink	pg_free_remain_segment	gs_fault_inject	gs_repair_file	local_clear_bad_block_info
gs_repair_page	-	-	-	-	-

audit_copy_exec

Parameter description: Specifies whether to audit the COPY operation.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer, **0** or **1**

- **0** indicates that auditing the COPY operation is disabled.
- **1** indicates that auditing the COPY operation is enabled.

Default value: **1**

audit_set_parameter

Parameter description: Specifies whether to audit the SET operation.

Parameter type: integer

Unit: none

Value range: **0** or **1**

- **0** indicates that auditing the SET operation is disabled.
- **1** indicates that auditing the SET operation is enabled.

Default value: **0**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

audit_xid_info

Parameter description: Specifies whether to record the transaction ID of the SQL statement in the **detail_info** column of the audit log.

Parameter type: integer

Unit: none

Value range: **0** or **1**

- **0** indicates that the function of recording transaction IDs in the audit log is disabled.
- **1** indicates that the function of recording transaction IDs in the audit log is enabled.

Default value: **0**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

NOTICE

If this function is enabled, the **detail_info** information in the audit log starts with *xid*. For example:

```
detail_info: xid=14619 , create table t1(id int);
```

If transaction IDs do not exist, **xid=NA** is recorded.

enableSeparationOfDuty

Parameter description: Specifies whether the separation of three duties is enabled.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the separation of duties is enabled.
- **off** indicates that the separation of duties is disabled.

Default value: **off**

enable_nonsysadmin_execute_direct

Parameter description: Specifies whether non-system administrators and non-monitor administrator are allowed to execute the EXECUTE DIRECT ON statement.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that any user is allowed to execute the EXECUTE DIRECT ON statement.
- **off** indicates that only the system administrator and monitor administrator are allowed to execute the EXECUTE DIRECT ON statement.

Default value: **off**

enable_access_server_directory

Parameter description: Specifies whether to allow non-initial users to create, modify, and delete directories.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that non-initial users have the permission to create, modify, and delete directories.
- **off** indicates that non-initial users do not have the permission to create, modify, and delete directories.

Default value: **off**

NOTICE

- For security purposes, only the initial user can create, modify, and delete DIRECTORY objects by default.
- If **enable_access_server_directory** is enabled, users with the SYSADMIN permission and users who inherit the gs_role_directory_create permission of the built-in role can create directories. A user with the SYSADMIN permission, the owner of a directory, a user who is granted with the DROP permission for the directory, or a user who inherits the gs_role_directory_drop permission of the built-in role can delete the directory. A user with the SYSADMIN permission and the owner of a directory object can change the owner of the directory, and the user must be a member of the new owning role.

14.3.23 CM Parameters

Modifying CM parameters affects the running mechanism of GaussDB. You are advised to ask GaussDB engineers to do it for you. For details about how to modify the CM parameters, see method 1 in **Table 14-2**.

14.3.23.1 CM Agent Parameters

log_dir

Parameter description: Specifies the directory where CM Agent logs are stored. The value can be an absolute path, or a path relative to the CM Agent data directory.

Value range: a string. Any modification of this parameter takes effect only after CM Agent is restarted. For details about how to modify this parameter, see **Table 14-2**.

Default value: **"log"**, indicating that CM Agent logs are generated in the CM Agent data directory.

log_file_size

Parameter description: Specifies the size of a log file. If a log file exceeds the specified size, a new one is created to record log information.

Value range: an integer ranging from 0 to 2047. The unit is MB. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **16MB**

log_min_messages

Parameter description: Specifies which message levels are written to the CM Agent log. Each level covers all the levels following it. The lower the level is, the fewer messages will be written into the log.

Value range: enumerated type. Valid values are **debug5**, **debug1**, **warning**, **error**, **log**, and **fatal**. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **warning**

incremental_build

Parameter description: Specifies whether a standby DN is incrementally built. If this parameter is enabled, a standby DN is incrementally built.

Value range: Boolean. The value can be **on** or **off**. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **on**

alarm_component

Parameter description: Specifies the location of the alarm component that processes alarms.

Value range: a string. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

- If **--alarm-type** in the gs_preinstall script is set to **5**, no third-party component is connected and alarms are written into system_alarm logs. In this case, the value of **alarm_component** is **/opt/huawei/snas/bin/snas_cm_cmd**.
- If **--alarm-type** in the gs_preinstall script is set to **1**, a third-party component is connected. In this case, the value of **alarm_component** is the absolute path of the executable program of the third-party component.

Default value: **/opt/huawei/snas/bin/snas_cm_cmd**

alarm_report_interval

Parameter description: Specifies the interval at which an alarm is reported. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Value range: a non-negative integer. The unit is s.

Default value: **1**

alarm_report_max_count

Parameter description: Specifies the maximum number of times an alarm is reported. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Value range: a non-negative integer

Default value: **1**

agent_report_interval

Parameter description: Specifies the interval at which CM Agent reports the instance status.

Value range: an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **1**

agent_phony_dead_check_interval

Parameter description: Specifies the interval at which CM Agent checks whether the DN process is suspended.

Value range: an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **10**

agent_check_interval

Parameter description: Specifies the interval at which CM Agent queries the status of instances, such as the DNs.

Value range: an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **2**

agent_heartbeat_timeout

Parameter description: Specifies the heartbeat timeout interval for CM Agent to connect to CM Server.

Value range: an integer ranging from 2 to 2^{31} – 1. The unit is second. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **8**

agent_connect_timeout

Parameter description: Specifies the time to wait before the attempt of CM Agent to connect to CM Server times out.

Value range: an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **1**

agent_connect_retries

Parameter description: Specifies the number of times CM Agent tries to connect to the CM Server.

Value range: an integer. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **15**

agent_kill_instance_timeout

Parameter description: Specifies the interval from the time when CM Agent fails to connect to the primary CM Server to the time when CM Agent terminates all instances on the node.

Value range: an integer. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **0**, indicating that the operation of stopping all instances on the node is not initiated.

security_mode

Parameter description: Specifies whether DNs are started in secure mode. If this parameter is enabled, DNs are started in secure mode. Otherwise, DNs are started in non-secure mode.

Value range: Boolean. The value can be **on** or **off**. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **off**

upgrade_from

Parameter description: Specifies the internal version number of the database before an in-place upgrade. Do not modify the value of this parameter.

Value range: a non-negative integer. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **0**

process_cpu_affinity

Parameter description: Specifies whether to bind a primary DN process to a CPU core before starting the process. If this parameter is set to **0**, core binding is not performed. If it is set to another value, core binding is performed, and the number of physical CPU cores is 2^n. Only Arm is supported.

Value range: an integer ranging from 0 to 2. The modification of this parameter takes effect only after the database and CM Agent are restarted. For details about how to modify the parameter, see **Table 14-2**.

Default value: **0**

log_threshold_check_interval

Parameter description: Specifies the interval for compressing and clearing logs.

Parameter type: integer

Unit: second

Value range: 0 to 2147483647

Default value: **1800**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. You can adjust the value based on the disk space and scenario requirements.

dilatation_shard_count_for_disk_capacity_alarm

Parameter description: Specifies the number of shards to be added in the scale-out scenario. This parameter is used to calculate the threshold for reporting a disk capacity alarm.

NOTE

The parameter value must be the same as the actual number of shards to be added.

Value range: an integer ranging from 0 to 2^{32} – 1. If this parameter is set to **0**, the disk scale-out alarm is not reported. If this parameter is set to a value greater than **0**, the disk scale-out alarm is reported and the threshold is calculated based on the number of shards specified by this parameter. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **1**

log_max_size

Parameter description: Specifies the maximum size of a log file.

Parameter type: integer

Unit: MB

Value range: 0 to 2147483647

Default value: **10240**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. You can adjust the value based on the disk space and scenario requirements.

log_max_count

Parameter description: Specifies the maximum number of logs that can be stored on hard disks.

Parameter type: integer

Unit: none

Value range: 0 to 10000

Default value: **10000**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. You can adjust the value based on the disk space and scenario requirements.

log_saved_days

Parameter description: Specifies the number of days for storing logs.

Parameter type: integer

Unit: day

Value range: 0 to 1000

Default value: **90**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. You can adjust the value based on the disk space and scenario requirements.

enable_log_compress

Parameter description: Specifies whether to enable log compression.

Value range: Boolean. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

- **on** indicates that log compression is enabled.
- **off** indicates that log compression is disabled.

Default value: **on**

agent_backup_open

Parameter description: Specifies the DR database instance settings. After this parameter is enabled, CM runs in DR database instance mode.

Value range: an integer ranging from 0 to 1. Any modification of this parameter takes effect only after CM Agent is restarted. For details about how to modify this parameter, see **Table 14-1**.

- **0**: disabled.
- **1**: enabled.

Default value: **0**

enable_xc_maintenance_mode

Parameter description: Specifies whether the pgxc_node system catalog can be modified when the database instance is in read-only mode.

Value range: Boolean. Any modification of this parameter takes effect only after CM Agent is restarted. For details about how to modify this parameter, see **Table 14-1**.

- **on** indicates that the pgxc_node system catalog can be modified.
- **off** indicates that the pgxc_node system catalog cannot be modified.

Default value: **on**

unix_socket_directory

Parameter description: Specifies the directory location of the Unix socket.

Value range: a string. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-1**.

Default value: ''

enable_dcf

Parameter description: Specifies the status of the DCF mode.

Value range: Boolean. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-1**.

- **0**: disabled.
- **1**: enabled.

Default value: **off**

disaster_recovery_type

Parameter description: Specifies the type of the DR relationship between the primary and standby database instances.

Value range: an integer ranging from 0 to 2. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-1**.

- **0** indicates that no DR relationship is established.
- **1** indicates that the OBS DR relationship is established.
- **2** indicates that the streaming DR relationship is established.

Default value: **0**

enable_e2e_rto

Parameter description: Specifies whether to enable the E2E RTO function. After this function is enabled, the hang-up detection period and network detection timeout interval are shortened. The CM can reach the E2E RTO indicator (RTO for a single instance ≤ 10s; RTO for combined faults ≤ 30s).

Parameter type: integer

Unit: none

Value range:

- **1**: enabled.
- **0**: disabled.

Default value:

14.3.23.2 CM Server Parameters

log_dir

Parameter description: Specifies the directory where CM Server logs are stored. The value can be an absolute path, or a path relative to the CM Server data directory.

Value range: a string. The modification takes effect after CM Server is restarted. For details about how to modify this parameter, see **Table 14-2**.

Default value: **"log"**, indicating that CM Server logs are generated in the CM Server data directory.

log_file_size

Parameter description: Specifies the size of a log file. If a log file exceeds the specified size, a new one is created to record log information.

Value range: an integer ranging from 0 to 2047. The unit is MB. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **16MB**

log_min_messages

Parameter description: Specifies which message levels are written to the CM Server log. Each level covers all the levels following it. The lower the level is, the fewer messages will be written into the log.

Value range: enumerated type. Valid values are **debug5**, **debug1**, **log**, **warning**, **error**, and **fatal**. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **warning**

thread_count

Parameter description: Specifies the number of threads in the agent thread pool.

Parameter type: string

Unit: none

Value range: This parameter consists of two parts: '**worker_thread_count** and **io_ratio**. The meanings of the two parts are as follows:

- **worker_thread_count**: specifies the number of AgentWorker threads. The value ranges from 2 to 1000.
- **io_ratio**: specifies the ratio of I/O threads to worker threads. The value *n* indicates that one AgentIO thread corresponds to *n* AgentWorker threads. The value ranges from 1 to 100. At least one AgentIO thread is required.

Default value: **"(1000,1)"**

Setting method: Refer to the **set cm parameters** table in "Unified Database Management Tool > cm_ctl Tool Introduction" in *Tool Reference*. Then run the reload command for the settings to take effect.

Setting suggestion: Retain the default value. If you want to adjust the value, make sure you understand the parameter meaning and adjust the value with caution to avoid risks caused by misoperations.

alarm_component

Parameter description: Specifies the location of the alarm component that processes alarms.

Value range: a string. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

- If **--alarm-type** in the gs_preinstall script is set to **5**, no third-party component is connected and alarms are written into system_alarm logs. In this case, the value of **alarm_component** is **/opt/huawei/snas/bin/snas_cm_cmd**.
- If **--alarm-type** in the **gs_preinstall** script is set to **1**, a third-party component is connected. In this case, the value of **alarm_component** is the absolute path of the executable program of the third-party component.

Default value: **/opt/huawei/snas/bin/snas_cm_cmd**

instance_failover_delay_timeout

Parameter description: Specifies the delay in CM Server failover after the primary CM Server breakdown is detected.

Value range: an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **0**

instance_heartbeat_timeout

Parameter description: Specifies the time to wait before the instance heartbeat times out.

Value range: an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **6**

cmserver_ha_connect_timeout

Parameter description: Specifies the time to wait before the connection between the primary and standby CM Servers times out.

Value range: an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: 2

cmserver_ha_heartbeat_timeout

Parameter description: Specifies the time to wait before the heartbeat between the primary and standby CM Servers times out.

Value range: an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **6**

phony_dead_effective_time

Parameter description: Specifies the maximum number of times DN processes are detected as zombie. If the number of times a process is detected as zombie is greater than the specified value, the process is considered to be a zombie process and will be restarted.

Value range: an integer. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **5**

enable_transaction_read_only

Parameter description: Specifies whether the database is in read-only mode.

Value range: Boolean values **on**, **off**, **true**, **false**, **yes**, **no**, **1**, and **0**. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **on**

datastorage_threshold_check_interval

Parameter description: Specifies the interval for checking the disk usage. The system checks the disk usage at the interval specified by the user.

Value range: an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **10**

datastorage_threshold_value_check

Parameter description: Specifies the usage threshold of a read-only disk in a database. When the disk usage of the data directory exceeds the specified value, the database is automatically set to read-only mode. In the read-only mode, log replay cannot be restricted. Pay attention to the cluster disk capacity and handle read-only alarms in a timely manner to prevent the disk space from being used up; otherwise, the database cannot be quickly restored.

Parameter type: integer

Unit: percentage

Value range: 1 to 99.

Default value: **85**

Setting method: The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Setting suggestion: Retain the default value. You can adjust this parameter based on the disk space. When adjusting this parameter, you are advised to adjust the **max_size_for_xlog_retention** parameter of the DN at the same time to prevent the instance read-only threshold from being triggered by backup operations.

max_datastorage_threshold_check

Parameter description: Specifies the maximum interval for checking the disk usage. After you modify the read-only mode parameter, the system automatically checks whether the disk usage reaches the threshold at the specified interval.

Value range: an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **43200**

cmserver_ha_status_interval

Parameter description: Specifies the interval between synchronizations of primary and standby CM Server status.

Value range: an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **1**

cmserver_self_vote_timeout

Parameter description: Specifies the time to wait before the CM Server self-voting times out.

Value range: an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **6**

alarm_report_interval

Parameter description: Specifies the interval at which an alarm is reported.

Value range: a non-negative integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **3**

alarm_report_max_count

Parameter description: Specifies the maximum number of times an alarm is reported.

Value range: a non-negative integer. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **1**

enable_az_auto_switchover

Parameter description: Specifies whether to enable automatic AZ switchover. If this parameter is enabled, CM Server automatically switches over services among AZs. Otherwise, when a DN is faulty, services will not be automatically switched to another AZ even if the current AZ is unavailable. You can run the switchover command to manually switch services to another AZ.

Value range: a non-negative integer. **0**: disabled; **1**: enabled. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **1**

instance_keep_heartbeat_timeout

Parameter description: The CM Agent periodically checks the instance status and reports the status to the CM Server. If the instance status cannot be detected for a long time and the accumulated number of times exceeds the value of this parameter, the CM Server delivers a command to the CM Agent to restart the instance.

Value range: an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **40**

az_switchover_threshold

Parameter description: If the failure rate of a DN shard in an AZ (Number of faulty DN shards/Total number of DN shards x 100%) exceeds the specified value, automatic AZ switchover is triggered.

Value range: an integer ranging from 0 to 100. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **100**

az_check_and_arbitrate_interval

Parameter description: Specifies the interval for checking the AZ status. If an AZ is abnormal, automatic AZ switchover is triggered.

Value range: an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: 2

az_connect_check_interval

Parameter description: Specifies the interval at which the network connection between AZs is checked.

Value range: an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **60**

az_connect_check_delay_time

Parameter description: Specifies the delay between two retries to check the network connection between AZs.

Value range: an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **150**

cmserver_demote_delay_on_etcd_fault

Parameter description: Specifies the interval at which CM Server switches from the primary state to the standby state due to unhealthy ETCD.

Value range: an integer. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **8**

instance_phony_dead_restart_interval

Parameter description: Specifies the interval at which the CM Agent process restarts and kills a zombie DN. The interval between two consecutive kill operations cannot be less than the value of this parameter. Otherwise, the CM Agent process does not deliver commands.

Value range: an integer. The unit is s. The minimum value that takes effect is **1800**. If this parameter is set to a value less than **1800**, the value **1800** takes effect. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: 21600

cm_auth_method

Parameter description: Specifies the port authentication mode of the CM. **trust** indicates that port authentication is not configured. **gss** indicates that Kerberos

port authentication is used. Note that you can change the value to **gss** only after the Kerberos server and client are successfully installed. Otherwise, the CM cannot communicate properly, affecting the database status.

Value range: **gss** or **trust**. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **trust**

cm_krb_server_keyfile

Parameter description: Specifies the location of the key file on the Kerberos server. The value must be an absolute path. The file is usually stored in the *${GAUSSHOME}*/**kerberos** directory and ends with keytab. The file name is the same as the name of the user who runs the database. This parameter is used together with **cm_auth_method**. If the **cm_auth_method** parameter is changed to **gss**, **cm_krb_server_keyfile** must also be configured as the correct path. Otherwise, the database status will be affected.

Value range: a string. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: *${GAUSSHOME}*/**kerberos/***{UserName}*.**keytab**. The default value cannot take effect and is used only as a prompt.

cm_server_arbitrate_delay_base_time_out

Parameter description: Specifies the basic delay duration for CM Server arbitration. If the primary CM Server is disconnected, the arbitration starts to be timed. If the disconnection duration exceeds the arbitration delay duration, a new primary CM Server will be selected. The arbitration delay duration is determined by the basic delay duration, the node index (server ID), and the incremental delay duration. The formula is as follows: Arbitration delay duration = Basic delay duration + Node index x Incremental delay duration

Value range: an integer. The unit is s. The index should be larger than 0. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: 10

cm_server_arbitrate_delay_incrememtal_time_out

Parameter description: Specifies the incremental delay duration for CM Server arbitration. If the primary CM Server is disconnected, the arbitration starts to be timed. If the disconnection duration exceeds the arbitration delay duration, a new primary CM Server will be selected. The arbitration delay duration is determined by the basic delay duration, the node index (server ID), and the incremental delay duration. The formula is as follows: Arbitration delay duration = Basic delay duration + Node index x Incremental delay duration

Value range: an integer. The unit is s. The index should be larger than 0. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **3**

force_promote

Parameter description: Specifies whether CM Server enables forcible switchover (that is, when the cluster status is unknown, the basic functions of the cluster are available at the cost of data loss). The value **0** indicates that forcible startup is disabled, and the value **1** indicates that forcible startup is enabled. This parameter applies to DNs.

Value range: an integer, **0** or **1**. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **0**

switch_rto

Parameter description: Specifies the delay for the forcible switchover of CM Server. When **force_promote** is set to **1** and a shard in the database does not have primary CM Server, the system starts timing. After the delay, forcible switchover is executed.

Value range: an integer ranging from 60 to 2147483647. The unit is s. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **600**

enable_finishredo_retrieve

Parameter description: Specifies whether to retrieve data of Xlogs that have been cut off by redo after a forcible CM Server switchover. If this parameter is set to **on**, data is automatically retrieved after a forcible switchover.

Parameter type: Boolean

Unit: none

Value range:

- **off**: disabled.
- **on**: enabled.

Default value: **off**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: If you can accept the loss of some data caused by forcible service (system) switchover, set this parameter to **on** when the service recovery has the highest priority.

backup_open

Parameter description: Specifies the DR database instance settings. After this parameter is enabled, CM runs in DR database instance mode.

Value range: an integer ranging from 0 to 1. The modification takes effect after CM Server is restarted. This parameter cannot be enabled for a non-DR database instance. For details about how to modify this parameter, see **Table 14-2**.

- **0**: disabled.
- **1**: enabled.

Default value: **0**

enable_dcf

Parameter description: Specifies the status of the DCF mode.

Value range: Boolean. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

- **0**: disabled.
- **1**: enabled.

Default value: **off**

install_type

Parameter description: Specifies the settings related to the DR database instance. It is used to distinguish whether the database instance is based on Dorado.

Value range: an integer ranging from 0 to 2. The modification takes effect after CM Server is restarted. This parameter cannot be enabled for a non-DR database instance. For details about how to modify this parameter, see **Table 14-2**.

- **0** indicates the database instance for which the DR relationship is not established.
- **1** indicates a Dorado-based database instance.
- **2** indicates a streaming-based database instance.

Default value: **0**

enable_ssl

Parameter description: Specifies whether to enable SSL.

Value range: Boolean. After this function is enabled, the SSL certificate is used to encrypt communication. The modification takes effect after CM Server is restarted. For details about how to modify this parameter, see **Table 14-2**.

- **on** indicates that SSL is enabled.
- **off** indicates that SSL is disabled.
- **Default value**: **off**

NOTICE

To ensure security, you are advised not to disable it. After this function is disabled, the CM does not use encrypted communication and all information is transmitted in plaintext, which may bring security risks such as eavesdropping, tampering, and spoofing.

ssl_cert_expire_alert_threshold

Parameter description: Specifies the SSL certificate expiration alarm time.

Value range: an integer. The unit is day. If the certificate expiration time is less than the value of this parameter, an alarm indicating that the certificate is about to expire is reported. The modification takes effect after CM Server is restarted. For details about how to modify this parameter, see **Table 14-2**.

Default value: **90**

ssl_cert_expire_check_interval

Parameter description: Specifies the period for checking whether the SSL certificate expires.

Value range: an integer. The unit is s. The modification takes effect after CM Server is restarted. For details about how to modify this parameter, see **Table 14-2**.

Default value: **86400**

delay_arbitrate_timeout

Parameter description: Specifies the waiting time for a node in the same AZ as the primary DN to be promoted to primary after redo replay.

Value range: an integer, in the range [0,21474836] (unit: second). The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **0**

ddb_type

Parameter description: Specifies whether to switch between ETCD and DCC modes.

Value range: an integer. **0**: ETCD; **1**: DCC. The modification takes effect after CM Server is restarted. For details about how to modify this parameter, see **Table 14-2**.

Default value: **0**

ddb_log_level

Parameter description: Sets the DDB log level.

To disable the log function, set this parameter to **"NONE"**, which cannot be used together with the following log levels:

To enable the log function, set this parameter to one or a combination of the following log levels: **"RUN_ERR|RUN_WAR|RUN_INF|DEBUG_ERR|DEBUG_WAR|DEBUG_INF|TRACE|PROFILE|OPER"**. If two or more log levels are used together, separate them with vertical bars (|). The log level cannot be set to an empty string.

Value range: a string containing one or a combination of the following log levels: **RUN_ERR|RUN_WAR|RUN_INF|DEBUG_ERR|DEBUG_WAR|DEBUG_INF|TRACE|PROFILE|OPER**. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **RUN_ERR|RUN_WAR|DEBUG_ERR|OPER|RUN_INF|PROFILE**

ddb_log_backup_file_count

Parameter description: Specifies the maximum number of log files that can be saved.

Value range: an integer ranging from 1 to 100. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **10**

ddb_max_log_file_size

Parameter description: Specifies the maximum number of bytes in a log.

Value range: a string, in the range [1M,1000M]. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **10M**

ddb_log_suppress_enable

Parameter description: Specifies whether to enable the log suppression function.

Value range: an integer. **0**: disabled; **1**: enabled. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **1**

ddb_election_timeout

Parameter description: Specifies the DCC election timeout period.

Value range: an integer, in the range [1,600], in seconds. The modification of this parameter takes effect after reloading. For details about how to modify the parameter, see **Table 14-2**.

Default value: **3**

enable_synclist_single_inst

Parameter description: Specifies whether to reduce copies to one primary and zero standby.

Parameter type: Boolean

Unit: none

Value range:

- **off**: This function is disabled.
- **on**: This function is enabled.

Default value: **off**

Setting method: Refer to the **set cm parameters** table in "Unified Database Management Tool > cm_ctl Tool Introduction" in *Tool Reference*. Then run the reload command for the settings to take effect.

Setting suggestion: Retain the default value.

enable_e2e_rto

Parameter description: Specifies whether to enable the E2E RTO function. After this function is enabled, the hang-up detection period and network detection timeout interval are shortened. The CM can reach the E2E RTO indicator (RTO for a single instance ≤ 10s; RTO for combined faults ≤ 30s).

Parameter type: integer

Unit: none

Value range:

- **1**: enabled.
- **0**: disabled.

Default value:

14.3.24 Upgrade Parameters

IsInplaceUpgrade

Parameter description: Specifies whether an upgrade is ongoing. This parameter cannot be modified by users. Only the sysadmin user can access this parameter.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates an upgrade is ongoing.
- **off** indicates no upgrade is ongoing.

Default value: **off**

inplace_upgrade_next_system_object_oids

Parameter description: Indicates the OID of a new system object during the in-place upgrade. The value of this parameter cannot be changed.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string

Default value: empty

upgrade_mode

Parameter description: Specifies the upgrade mode.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

Value range: an integer ranging from 0 to *INT_MAX*

- **0** indicates that no upgrade is ongoing.
- **1** indicates that a local upgrade is ongoing.
- **2** indicates that a grayscale upgrade is ongoing.

Default value: **0**

NOTE

Special case: When the gray upgrade is used, if the major version upgrade policy is selected, that is, the upgrade script needs to be executed and the binary package needs to be replaced, the value of **upgrade_mode** is set to **2**;iIf the minor version upgrade policy is selected, that is, only the binary package needs to be replaced, the value of **upgrade_mode** is not set to **2**.

14.3.25 Miscellaneous Parameters

enable_default_ustore_table

Parameter description: Specifies whether to enable the Ustore by default. If this parameter is set to **on**, all created tables are Ustore tables.

Parameter type: Boolean.

Value range:

- **on** indicates that the Ustore is enabled by default.
- **off** indicates that the Ustore is disabled by default.

Default value: **on**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**. Note that the **track_counts** and **track_activities** parameters must be enabled when the Ustore table is used. Otherwise, space bloat may occur.

enable_ustore

Parameter description: Specifies whether to enable the Ustore. If this parameter is set to **on**, Ustore tables can be created. Note that the **track_counts** and **track_activities** parameters must be enabled when the Ustore table is used. Otherwise, space bloat may occur.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: [off,on]

Default value: **on**

enable_segment_datafile_preallocate

Parameter description: Specifies whether to allocate disk space immediately during segment-page file extension. If this parameter is set to **on**, the system preferentially uses fallocate to allocate disk space immediately during file expansion. If the system does not support fallocate, the system applies for disk space by writing zeros byte by byte. If this parameter is set to **off**, the system does not allocate disk space immediately during file expansion and hole files are created.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: [off,on]

Default value: **on**

reserve_space_for_nullable_atts

Parameter description: Specifies whether to reserve space for the nullable attribute of an Ustore table. If this parameter is set to **on**, space is reserved for the nullable attribute of the Ustore table by default.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: [off,on]

Default value: **on**

server_version

Parameter description: Specifies the server version number.

This is a fixed parameter of the INTERNAL type. It can be viewed but cannot be modified. This parameter is not recommended. To query the server version, use the opengauss_version() function.

Value range: a string.

Default value: **9.2.4**

server_version_num

Parameter description: Specifies the server version number.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

Value range: an integer

Default value: **90204**

block_size

Parameter description: Specifies the block size of the current database.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

Value: **8192**

Default value: **8192**

segment_size

Parameter description: Specifies the segment file size of the current database.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

Unit: 8 KB

Default value: **131072**, that is, 1 GB.

max_index_keys

Parameter description: Specifies the maximum number of index keys supported by the current database.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

Default value: **32**

integer_datetimes

Parameter description: Specifies whether the date and time are in the 64-bit integer format.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

Value range: Boolean

- **on**: yes.
- **off**: no.

Default value: **on**

lc_collate

Parameter description: Specifies the locale in which sorting of textual data is done.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

Default value: Determined by the configuration during the database installation and deployment.

lc_ctype

Parameter description: Specifies the locale that determines character classifications. For example, it specifies what a letter and its upper-case equivalent are.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

Default value: Determined by the configuration during the database installation and deployment.

max_identifier_length

Parameter description: Specifies the maximum identifier length.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

Value range: an integer.

Default value: **63**

server_encoding

Parameter description: Specifies the database encoding (character set).

By default, gs_initdb will initialize the setting of this parameter based on the current system environment. You can also run the **locale** command to check the current configuration environment.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

Default value: determined by the current system environment when the database is created.

basebackup_timeout

Parameter description: Specifies the timeout interval for a connection that has no read or write operations after a backup transfer is complete.

When the gs_basebackup tool is used for transmission and a high compression rate is specified, the transmission of the tablespace may time out (the client needs to compress the transmitted data).

Value range: an integer ranging from 0 to *INT_MAX*. The unit is s. **0** indicates that archiving timeout is disabled.

Default value: 600s

datanode_heartbeat_interval

Parameter description: Specifies the interval at which heartbeat messages are sent between heartbeat threads. You are advised to set this parameter to a value no more than **wal_receiver_timeout**/2.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1000 to 60000. The unit is ms.

Default value: **1s**

max_concurrent_autonomous_transactions

Parameter description: Specifies the maximum number of autonomous transaction connections, that is, the maximum number of concurrent autonomous transactions executed at the same time. If this parameter is set to **0**, autonomous transactions cannot be executed.

Parameter type: integer.

Unit: none

Value range: 0–10000. The theoretical maximum value is **10000**, and the actual maximum value is a dynamic value. The calculation formula is 262143 –

job_queue_processes – **autovacuum_max_workers** – **max_inner_tool_connections** – **max_connections** – **AUXILIARY_BACKENDS** – **AV_LAUNCHER_PROCS**. The values of **job_queue_processes**, **autovacuum_max_workers**, **max_inner_tool_connections**, and **max_connections** depend on the settings of the corresponding GUC parameters. **AUXILIARY_BACKENDS** indicates the number of reserved auxiliary threads, which is fixed to **20**. **AV_LAUNCHER_PROCS** indicates the number of reserved launcher threads for autovacuum, which is fixed to **2**.

Default value: **200** (196-core CPU/1536 GB memory, 128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory); **150** (96-core CPU/768 GB memory); **120** (80-core CPU/640 GB memory) **100** (64-core CPU/512 GB memory); **80** (60-core CPU/480 GB memory); **40** (32-core CPU/256 GB memory); **20** (16-core CPU/128 GB memory); **10** (8-core CPU/64 GB memory, 4-core CPU/32 GB memory, 4-core CPU/16 GB memory)

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Suggestion: Set this parameter based on actual service requirements and hardware configurations. It is recommended that this parameter be set to a value less than or equal to 1/10 of **max_connections**. If you only increase the value of this parameter but do not adjust the memory parameters in the same proportion, the memory may be insufficient and the error message "memory is temporarily unavailable" is displayed when the service load is heavy.

NOTE

If the value range of this parameter is changed during the upgrade and the value is changed before the commit operation, you need to change the value range to the value allowed before the upgrade if you roll back the upgrade. Otherwise, the database may fail to be started.

enable_seqscan_fusion

Parameter description: Specifies whether to enable SeqScan optimization.

This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that SeqScan optimization is enabled.
- **off** indicates that SeqScan optimization is disabled.

Default value: **off**

NOTE

This parameter can be used to optimize only the execution time of the SeqScan operator in the EXPLAIN ANALYZE statement.

cluster_run_mode

Parameter description: Specifies whether a DN belongs to the primary or standby database instance in the dual-database instance DR scenario. For a single database instance, use the default primary database instance.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: enumerated values

- **cluster_primary** indicates the primary database instance.
- **cluster_standby** indicates the standby database instance.

Default value: **cluster_primary**

acceleration_with_compute_pool

Parameter description: Determines whether to use the computing resource pool for acceleration when an OBS is queried. (Due to specification changes, the current version no longer supports this feature. Do not use it.)

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the query covering the OBS is accelerated based on the cost when the computing resource pool is available.
- **off** indicates that no query is accelerated using the computing resource pool.

Default value: **off**

dfs_partition_directory_length

Parameter description: Specifies the maximum directory name length for the partition directory of a table partitioned by VALUE in the HDFS.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: 92 to 7999

Default value: **512**

max_resource_package

Parameter description: Specifies the maximum number of threads that each DN can run concurrently on an acceleration database instance on the cloud.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: 0 to 2147483647

Default value: **0**

enable_gpi_auto_update

Parameter description: Determines whether global indexes are updated by default in partition DDL commands.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- The value **on** indicates that global indexes are updated regardless of whether the partition DDL commands contain the UPDATE GLOBAL INDEX clause.
- The value **off** indicates that global indexes are not updated unless the partition DDL commands contain the UPDATE GLOBAL INDEX clause.

Default value: **off**

enable_gsplsql_execopt

Parameter description: Determines whether to optimize the execution of stored procedure functions in Tuple mode.

Execution modes of stored procedure functions are as follows:

- Statement mode: Use the perform or call procedure() function to call stored procedures, during which every SQL statement in the stored procedures is executed separately.
- Tuple mode: Use **targetlist** or **trigger** to call stored procedures. The calling environment is one tuple input in the execution state.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that when a stored procedure function is cyclically executed in Tuple mode, the information that does not change during the initialization of the stored procedure is allocated and recorded to avoid invalid repeated calling during the execution.
- **off** indicates that the default calling logic is used and information is initialized in each loop.

Default value: **on**

multi_insert_min_rows

Parameter description: Specifies the minimum estimated number of rows to be inserted in batches. You can run **explain (verbose on)** to check whether the execution plan contains "Batch Insert."

Parameter type: integer.

Unit: none

Value range: –1 to 2147483647. **–1** indicates that batch insertion is not used. If the value is greater than or equal to **0**, the estimated number of rows to be inserted is greater than or equal to **multi_insert_min_rows**, and batch insertion is used for the INSERT SELECT statement to improve performance.

Default value: **1000**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: You are advised to use the default value or a value greater than **1000**. If the number of inserted data rows is small, batch insertion may deteriorate the insertion performance.

NOTE

- **Specification**

  By default, the batch insertion rate of an Astore table can be improved by about 50%, and the batch insertion rate of a Ustore table can be improved by about 60%.

- **Constraint**

  In addition to the **multi_insert_min_rows** parameter constraint, batch insertion has the following constraints:

  1. Statements other than INSERT SELECT are not supported.
  2. Tables that contain BEFORE and INSTEAD triggers are not supported because the data to be inserted may be changed.
  3. SQL statements cannot contain the volatile function because a new transaction is started each time a volatile function is executed. However, if batch insertion is used, the snapshot obtained by the volatile function may be different from that obtained by common insertion.
  4. The SQL statement cannot contain RETURNING.
  5. The UPSERT clause is not supported.
  6. Batch insertion cannot be used for unlogged tables, temporary tables, or global temporary tables. Otherwise, the performance may deteriorate.
  7. Tables containing auto increment columns are supported only when **b_format_version** is set to **'5.7'** or **b_format_dev_version** is set to **'s2'**.

Example:

```
-- Create tables and data.
create table t(id int, c1 varchar(20), c2 varchar(40), c3 varchar(50)) with (storage_type=astore);
create table t_tmp(id int, c1 varchar(20), c2 varchar(40), c3 varchar(50))  with (storage_type=astore);
insert into t_tmp values (generate_series(1, 1000), '1111111111111111111', '222222222222222222',
'33333333333333333333333333333333');
analyze t_tmp;

-- Insert 1000 rows of data for 1000 times. Batch insertion takes 1.2s while non-batch insertion takes 2.68s.
Using batch insertion improves the performance by more than 50%.
set multi_insert_min_rows=-1;
DECLARE
    start_time bigint;
    end_time bigint;
BEGIN
    start_time := dbe_utility.get_time();
    for i in 1 .. 1000 loop
        insert into t select * from t_tmp;
    end loop;
    end_time := dbe_utility.get_time();
    dbe_output.print_line('Time consumed:' ||(end_time - start_time)/100);
END;
/
Time consumed: 2.68

set multi_insert_min_rows=0;
DECLARE
    start_time bigint;
    end_time bigint;
BEGIN
    start_time := dbe_utility.get_time();
    for i in 1 .. 1000 loop
        insert into t select * from t_tmp;
    end loop;
    end_time := dbe_utility.get_time();
    dbe_output.print_line('Time consumed:' ||(end_time - start_time)/100);
END;
/
Time consumed: 1.2
```


enable_force_smp

Parameter description: Specifies whether to forcibly enable SMP. When this parameter is enabled, the cost of starting the stream thread is empty by default. When the degree of parallelism is set and the operator supports parallelism, the parallel path is forcibly selected.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that SMP is forcibly enabled.
- **off** indicates that SMP is not forcibly enabled.

Default value: **off**

NOTICE

- This parameter is valid only for operators that support SMP. For details, contact the administrator.
- In addition, the constraints on forcibly enabling SMP are as follows:
  1. The SMP feature improves the performance through operator parallelism and occupies more system resources, including CPU, memory, and I/O. It is used to save time at the cost of resources. By setting parallelism, SMP improves system performance in appropriate scenarios when resources are sufficient.
  2. If the scenarios are inappropriate (for example, the data volume is small) or resources are insufficient, the performance may deteriorate.
  3. This parameter is not supported in scenarios where SMP is not applicable.

enable_partrouting_optimization

Parameter description: Specifies whether to optimize the insertion of partitioned tables.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the insertion of partitioned tables is optimized. For the INSERT SELECT statement, if the SELECT statement contains a constant partition key, partition routing needs to be performed only once for the INSERT statement, improving performance.
- **off** indicates that the insertion of partitioned tables is not optimized. Before inserting each piece of data, you need to perform partition routing to determine the partitioned table to be inserted.

Default value: **on**

NOTE

Constraints for optimizing the insertion of partitioned tables:

1. Only the INSERT SELECT statement is supported.
2. INSERT INTO ta SELECT FROM tb: For all partition key values inserted to table **a**, the column values in the corresponding SELECT result set must be constants. (This column is optional because the default values are constants.)

   (1) INSERT INTO ta SELECT c,d FROM b WHERE tb.c='1' or SELECT '1' as c, d FROM tb: The result column **tb.c** is a constant.

   (2) INSERT INTO ta SELECT c,d FROM b WHERE tb.c=func('1');: If func() is neither a volatile function nor a stable/immutable function that contains non-constant parameters, **tb.c** can be determined as a constant.

   (3) If the column value is the return value of an aggregate function, for example, INSERT INTO ta SELECT count(c),d FROM b WHERE tb.c='1' group by c, d;, the count(c) cannot be determined as a constant.
3. The UPSERT clause is not supported.
4. Tables that contain BEFORE and INSTEAD triggers are not supported because the data to be inserted may be changed.

14.3.26 Wait Events

enable_instr_track_wait

Parameter description: Specifies whether to enable real-time collection of wait event information.

In the x86-based centralized deployment scenario, the hardware configuration specifications are 32-core CPU and 256 GB memory. When the Benchmark SQL 5.0 tool is used to test performance, the performance fluctuates by about 1.4% by enabling or disabling this parameter.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the function of collecting wait event information is enabled.
- **off** indicates that the function of collecting wait event information is disabled.

Default value: **on**

14.3.27 Query

instr_unique_sql_count

Parameter description: Specifies the maximum number of unique SQL records to be collected. The value **0** indicates that the function of collecting unique SQL information is disabled.

If the value is changed from a larger one to a smaller one, the original data in the system will be cleared and re-collected (the standby node does not support this function). There is no impact if the value is changed from a smaller one to a larger one.

When the number of unique SQL records generated in the system (to view the statistics, query **dbe_perf.statement** or **dbe_perf.summary_statement**) is greater

than the value of **instr_unique_sql_count**, the extra unique SQL records are not collected.

In the x86-based centralized deployment scenario, the hardware configuration specifications are 32-core CPU and 256 GB memory. When the Benchmark SQL 5.0 tool is used to test performance, the performance fluctuates by about 3% by enabling or disabling this parameter.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 2147483647.

Default value: **200000**

instr_unique_sql_combination_options

Parameter description: Specifies the configuration items of combining unique SQL statements of the same type. The value of this parameter consists of multiple configuration items separated by commas (,).

If this feature is enabled, the IDs of unique SQL statements of the same type are normalized, and the generated unique SQL strings are normalized.

Parameter type: string.

Unit: none

Value range: See **Table 14-30**.

NOTICE

- When configuring the combination function, set the character string based on **Table 14-30**. Use commas (,) to separate multiple configuration items, for example, **set instr_unique_sql_combination_options='in_clause';**.
- If this parameter is left blank, this function is disabled, for example, **set instr_unique_sql_combination_options='';**.

Table 14-30 Configuration items of the combination function

Configuration Item	Behavior Control
in_clause	Combines only fixed parameters and precompiled binding parameters in the IN clause of the SELECT IN() statement. Example 1: **select * from table where column in (1,2,3);** Unique SQL string after combination: **select * from table where column in (1... n);** Example 2: **select * from table where column in ($1,$2,$3);** Unique SQL string after combination: **select * from table where column in (1... $n);** Example 3: **select * from example_table where column in (1,2,$1,3,$2);** Unique SQL string after combination: **select * from example_table where column in (1...n,$1...$n);** Example 4: **select * from example_table where (column1, column2) in ((1, 'a'), (2, 'b'), (3, 'c'));** Unique SQL string after combination: **select * from example_table where (column1, column2) in ((1...n));**

Default value: **'in_clause'**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: By default, this function is disabled during the upgrade and enabled when a new instance is delivered.

CAUTION

- If this feature is used, the unique SQL IDs of the query statements involved in combination are changed, which affects the SQL statements for which SQL patch has been created.
- For values of the bigint, real, float4, blob, numeric, decimal, number, dec, or integer type, if **in()** contains a single parameter or multiple parameters, two different **unique_sql_id** values are generated.

instr_unique_sql_track_type

Parameter description: Specifies which SQL statements are recorded in Unique SQL.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: enumerated values

- **top**: Only top-level SQL statements are recorded.
- **all**: All SQL statements are recorded.

Default value: **all**

NOTE

The default value of the parameter is **all**. Compared with **top**, if the parameter is set to **all**, additional statistics about statements in the stored procedure are recorded, which causes performance loss for the execution of the stored procedure and more severe impact on the performance of statements that are executed quickly, such as simple expressions in the stored procedure.

enable_instr_rt_percentile

Parameter description: Specifies whether to enable the function of calculating the response time of 80% and 95% SQL statements in the system.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the function is enabled.
- **off** indicates that the function is disabled.

Default value: **on**

percentile

Parameter description: Specifies the percentage of SQL statements whose response time is to be calculated by the background calculation thread.

This is an INTERNAL parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string.

Default value: **80,95**

instr_rt_percentile_interval

Parameter description: Specifies the interval at which the background calculation thread calculates the SQL response time.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 3600. The unit is s.

Default value: **10s**

enable_instr_cpu_timer

Parameter description: Specifies whether to capture the CPU time consumed during SQL statement execution.

In the x86-based centralized deployment scenario, the hardware configuration specifications are 32-core CPU and 256 GB memory. When the Benchmark SQL 5.0

tool is used to test performance, the performance fluctuates by about 3.5% by enabling or disabling this parameter.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the CPU time consumed during SQL statement execution is captured.
- **off** indicates that the CPU time consumed during SQL statement execution is not captured.

Default value: **on**

enable_stmt_track

Parameter description: Specifies whether to enable the full/slow SQL statement feature.

In the x86-based centralized deployment scenario, the hardware configuration specifications are 32-core CPU and 256 GB memory. When the Benchmark SQL 5.0 tool is used to test performance, the performance fluctuates by about 1.2% by enabling or disabling this parameter.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: Full/Slow SQL capture is enabled.
- **off**: Full /Slow SQL capture is disabled.

Default value: **on**

track_stmt_parameter

Parameter description: After **track_stmt_parameter** is enabled, the executed statements recorded in **statement_history** are not normalized. The complete SQL statement information can be displayed to help the database administrator locate faults. For a simple query, the complete statement information is displayed. For a PBE statement, the complete statement information and information about each variable value are displayed. The format is "query string; parameters: $1=value1,$2=value2, ..." This parameter is used to display complete SQL information and is not controlled by the **track_activity_query_size** parameter.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: The function of displaying complete SQL statement information is enabled.
- **off**: The function of displaying complete SQL statement information is disabled.

Default value: **off**

track_stmt_session_slot

Parameter description: Specifies the maximum number of full/slow SQL statements that can be cached in a session. If the number of full/slow SQL statements exceeds this value, new statements will not be traced until the flush thread flushes the cached statements to the disk to reserve idle space.

Parameter type: integer.

Unit: none

Value range: 0 to 2147483647.

Default value: 1000

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value, that is, the maximum number of full SQL slots that can be reserved for each session. If the value of this parameter is too large, a large amount of memory is occupied. If the value of this parameter is too small, full SQL statements may be lost.

track_stmt_details_size

Parameter description: Specifies the maximum size of execution events that can be collected by a single statement.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 100000000. The unit is byte.

Default value: **4096**

track_stmt_retention_time

Parameter description: Specifies the retention period of full/slow SQL statement records. This is a combination of parameters. This parameter is read every 60 seconds and records exceeding the retention period are deleted. Only the sysadmin user can access this parameter.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string consisting of two parts in the format of 'full sql retention time, slow sql retention time'.

- **full sql retention time** indicates the retention period of full SQL statements. The value ranges from 0 to 86400. The unit is second.
- **slow sql retention time** indicates the retention period of slow SQL statements. The value ranges from 0 to 604800. The unit is second.

Default value: **3600,604800**

track_stmt_stat_level

Parameter description: Determines the level of statement execution tracing.

Parameter type: character

Unit: none

Value range:

This parameter consists of two parts in the format of 'full sql stat level, slow sql stat level'.

The first part indicates the tracing level of full SQL statements. The value can be **OFF**, **L0**, **L1**, or **L2**.

The second part indicates the tracing level of slow SQL statements. The value can be **OFF**, **L0**, **L1**, or **L2**.

NOTE

If the tracing level of full SQL statements is not **OFF**, the current SQL statement tracing level is a higher level (L2 > L1 > L0) of full and slow SQL statements. For details about the levels, see "System Catalogs and System Views > System Catalogs > STATEMENT_HISTORY > STATEMENT_HISTORY columns" in *Developer Guide*.

Default value: **"OFF,L0"**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. If the full SQL tracing function is enabled, the performance is affected and a large amount of disk space may be occupied.

enable_auto_clean_unique_sql

Parameter description: Specifies whether to enable the automatic elimination function of unique SQL statements when the number of unique SQL statements generated in the system is greater than or equal to the value of **instr_unique_sql_count**.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: The automatic elimination function of unique SQL statements is enabled.
- **off**: The automatic elimination function of unique SQL statements is disabled.

Default value: **off**

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Set the value based on service requirements. If this function is enabled, certain disk space can be saved, but WDRs may be lost.

CAUTION

Some snapshot information comes from unique SQL statements. Therefore, when automatic elimination is enabled, if the selected start snapshot and end snapshot exceed the elimination time, the WDR report cannot be generated.

asp_log_directory

Parameter description: Specifies the directory for storing ASP log files on the server when **asp_flush_mode** is set to **all** or **file**. The value can be an absolute path, or relative to the **data** directory. Only the sysadmin user can access this parameter.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

If the value of **asp_log_directory** in the configuration file is an invalid path, the database instance cannot be restarted.

NOTE

- Valid path: Users have read and write permissions on the path.
- Invalid path: Users do not have read or write permission on the path.

Value range: a string

Default value: specified during installation.

enable_slow_query_log (Discarded)

Parameter description: Specifies whether to write the slow query information to the log file. This parameter is discarded in this version.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: indicates that slow query information needs to be written into log files.
- **off**: indicates that slow query information does not need to be written into log files.

Default value: **on**

query_log_file (Discarded)

Parameter description: Specifies the name of a slow query log file on the server. If **enable_slow_query_log** is set to **ON**, slow query records are written into log files. Only the sysadmin user can access this parameter. Generally, log file names are generated in strftime mode. Therefore, the system time can be used to define log file names, which are implemented using the escape character %. This function has been discarded in this version.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

You are advised to use the escape character % to specify the log file names for efficient management of log files.

Value range: a string

Default value: slow_query_log-%Y-%m-%d_%H%M%S.log

query_log_directory (Discarded)

Parameter description: Specifies the directory for storing low query log files when **enable_slow_query_log** is set to **on**. Only the sysadmin user can access this parameter. It can be an absolute path or a relative path (relative to the data directory), which has been discarded in this version.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

NOTICE

If the value of **query_log_directory** in the configuration file is an invalid path, the database instance cannot be restarted.

NOTE

Valid path: Users have read and write permissions on the path.

Invalid path: Users do not have read or write permission on the path.

Value range: a string

Default value: specified during installation.

perf_directory

Parameter description: Specifies the directory of the output file of the performance view dotting task. Only the sysadmin user can access this parameter. The value can be an absolute path, or relative to the data directory.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

NOTE

- Valid path: Users have read and write permissions on the path.
- Invalid path: Users do not have read or write permission on the path.

Value range: a string

Default value: specified during installation.

unique_sql_retention_time

Parameter description: Specifies the memory cleanup interval for the unique SQL hash table. The default value is 30 minutes.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to 3650. The unit is minute.

Default value: **30min**

track_stmt_standby_chain_size

Parameter description: Specifies the maximum memory and disk space occupied by fast/slow SQL statement records on the standby node. This is a combination of parameters. Only the SysAdmin user can access the database.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string

This parameter consists of four parts in the format of 'Full SQL memory size, Full SQL disk size, Slow SQL memory size, Slow SQL disk size'.

Full SQL and slow SQL statements are stored in different locations. Therefore, four additional values are used for control.

- **Full SQL memory size** indicates the maximum memory space reserved for fast SQL statements. The value range is [16,1024], in MB.
- **Full SQL disk size** indicates the maximum disk space occupied by reserved fast SQL statements. The value range is [512,1048576], in MB.
- **Slow SQL memory size** indicates the maximum memory space reserved for slow SQL statements. The value range is [16,1024], in MB.
- **Slow SQL disk size** indicates the maximum disk space reserved for slow SQL statements. The value range is [512,1048576], in MB.

The memory size cannot be greater than the disk size.

Default value: **32, 1024, 16, 512**

track_stmt_flush_mode

Parameter description: Specifies the storage mode of full SQL statements.

Parameter type: character

Unit: none

Value range:

This parameter consists of two parts in the format of 'full sql flush mode, slow sql flush mode':

- The first part indicates the full SQL tracing mode. The value can be **MEMORY** or **FILE**. If this parameter is set to **MEMORY**, full SQL statements are recorded in the memory. If this parameter is set to **FILE**, full SQL statements are recorded in disk files.
- The second part indicates the slow SQL tracing mode. In the current version, the value can only be **FILE**. If this parameter is set to **FILE**, slow SQL statements are recorded in disk files.

Default value: **"FILE,FILE"**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: If this parameter is set to **"MEMORY,FILE"**, the kernel supports full SQL statements, which occupies certain shared memory. The memory size is specified by the **track_stmt_shm_size** parameter.

track_stmt_shm_size

Parameter description: Specifies the size of the full SQL shared memory.

Parameter type: integer

Unit: byte

Value range: 134217728 to 1073741824

Default value: **134217728**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

14.3.28 System Performance Snapshot

enable_wdr_snapshot

Parameter description: Specifies whether to enable the database monitoring snapshot function.

Parameter type: Boolean

Unit: none

Value range:

- **on** indicates that the database monitoring snapshot function is enabled.
- **off** indicates that the database monitoring snapshot function is disabled.

Default value: **on**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

NOTE

You are advised not to enable it in the following scenarios:

- In multi-table and multi-database scenarios, WDR snapshot records snapshots in each database in serial mode, which takes a long time. Therefore, you are advised not to enable it.
- In the multi-table scenario, the query speed of the pg_stat_all_tables is slow due to the performance problems of some internal tables. In this case, the WDR snapshot function is slow. Therefore, you are advised not to enable it.
- If WDR snapshot is performed when there are a large number of DDL statements, WDR snapshot may fail. Therefore, you are advised not to enable it.

enable_wdr_snapshot_standby

Parameter description: Specifies whether to enable the database monitoring snapshot function on the standby node.

Parameter type: Boolean

Unit: none

Value range:

- **on** indicates that the database monitoring snapshot function is enabled on the standby node.
- **off** indicates that the database monitoring snapshot function is disabled on the standby node.

Default value: **off**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. If you want to adjust the value, make sure you understand the parameter meaning and adjust the value with caution to avoid risks caused by misoperations.

enable_show_standby_name

Parameter description: Specifies whether to display the names that distinguish the primary and standby nodes of the same shard. After this function is enabled, different names are returned when **dbe_perf.node_name** is queried on different nodes of the same shard.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: The function of distinguishing primary and standby node names is enabled.
- **off**: The function of distinguishing primary and standby node names is disabled.

Default value: **off**

wdr_snapshot_retention_days

Parameter description: Specifies the number of days database monitoring snapshots are retained. If the number of snapshots generated during database running exceeds the maximum number of snapshots that can be generated within the retention period (24 x 8 = 192 by default), the system deletes the snapshots with the smallest ID at an interval specified by the value of **wdr_snapshot_interval**.

NOTE

This parameter is valid only when **enable_wdr_snapshot** is set to **on**.

Parameter type: integer

Unit: day

Value range: 1 to 30

Default value: **8**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. A larger parameter value indicates higher disk usage.

wdr_snapshot_query_timeout

Parameter description: Specifies the execution timeout for the SQL statements associated with database monitoring snapshot operations. If the statement execution is not complete and no result is returned within the specified time, the snapshot operation fails.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**. The value **0** indicates that this parameter does not take effect.

Value range: an integer ranging from 0 to *INT_MAX*. The unit is second.

Default value: **100s**

wdr_snapshot_interval

Parameter description: Specifies the interval at which the backend thread snapshot automatically takes snapshots of the database monitoring data.

NOTE

This parameter is valid only when **enable_wdr_snapshot** is set to **on**.

Parameter type: integer

Unit: minute

Value range: 10 to 60

Default value: **60**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**. For example, if the value is **60** without a unit, **wdr_snapshot_interval** indicates 60 minutes. If the value is **1h**, **wdr_snapshot_interval** indicates 1 hour. If the unit is required, the value must be **min**, **h**, or **d**.

Setting suggestion: Retain the default value. If the retention period is fixed, a smaller value of this parameter indicates a larger disk usage.

wdr_snapshot_space_threshold

Parameter description: Specifies the threshold at which the space used by snapshots is controlled. When the space used by snapshots reaches 50% of the value of this parameter, the control logic of the database is enabled to stabilize the space used by the snapshots.

NOTE

- This parameter is valid only when **enable_wdr_snapshot** is set to **on**.
- If this parameter is set to a value smaller than the space used by the snapshots, the space usage does not shrink. Instead, the control logic is enabled to control the snapshot growth and stabilize the space used by the snapshots.

Parameter type: integer

Unit: KB

Value range: 0–107374182400

Default value: **0**, indicating that the function of controlling the space used by snapshots based on the space threshold is disabled.

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Set this parameter based on the actual scenario.

wdr_snapshot_full_backup_interval

Parameter description: Specifies the interval at which a full WDR snapshot is created. The interval specified by this parameter is about a number instead of time. For example, if the parameter is set to **12**, a full snapshot and then 11 incremental snapshots are generated for each group. If the parameter is set to **1**, all snapshots generated are full snapshots.

NOTE

- This parameter is valid only when **enable_wdr_snapshot** is set to **on**.
- If you change the value of this parameter after multiple snapshots have been generated, the next snapshot will be a full snapshot. For example, after five WDR incremental snapshots have been generated, if the parameter value is changed to **10**, a full snapshot will be generated before a new interval starts.
- When a snapshot is deleted, other snapshots in the same group are deleted. The number of snapshots to be deleted depends on the value of this parameter when the earliest group of snapshots are generated and is irrelevant to the current value.

Parameter type: integer

Unit: none

Value range: 1 to 24

Default value: **12**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Set this parameter based on the actual scenario. If this parameter is set to a smaller value, more full snapshots are generated, but fewer snapshots that can be retained with the same space used.

asp_flush_mode

Parameter description: Specifies the mode for the ASP to update data to the disk. The value can be **'file'** (default value), **'table'** (system catalog), or **'all'** (system catalog and file). Only the sysadmin user can access this parameter.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string, which can be **'table'**, **'file'**, or **'all'**

Default value: **'table'**

asp_flush_rate

Parameter description: When the number of samples reaches the value of **asp_sample_num**, the samples in the memory are updated to the disk based on a certain proportion. **asp_flush_rate** indicates the update proportion. If this parameter is set to **10**, it indicates that the update ratio is 10:1.

Parameter type: integer

Unit: none

Value range: 1 to 10

Default value: **10**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

asp_log_filename

Parameter description: Specifies the file name format when writing files using ASP. Only the sysadmin user can access this parameter.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string

Default value: **"asp-***%Y-%m-%d_%H%M%S***.log"**

asp_retention_days

Parameter description: Specifies the maximum number of days for reserving ASP samples when they are written to the system catalog.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to 7

Default value: **2**

asp_sample_interval

Parameter description: Specifies the sampling interval.

Parameter type: integer

Unit: second

Value range: 1 to 10

Default value: **1**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

asp_sample_num

Parameter description: Specifies the maximum number of samples allowed in the LOCAL_ACTIVE_SESSION view. Only the sysadmin user can access this parameter.

Parameter type: integer

Unit: none

Value range: 10 to 100000

Default value: **100000** (196-core CPU/1536 GB memory, 128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 64-core CPU/512 GB memory, 60-core CPU/480 GB memory, 32-core CPU/256 GB memory, 16-core CPU/128 GB memory, 8-core CPU/64 GB memory, 4-core CPU/32 GB memory); **36000** (4-core CPU/16 GB memory)

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. If you want to adjust the value, make sure you understand the parameter meaning and adjust the value with caution to avoid risks caused by misoperations.

enable_asp

Parameter description: Specifies whether to enable the active session profile function.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: The active session profile function is enabled.
- **off**: The active session profile function is disabled.

Default value: **on**

gs_perf_interval

Parameter description: Specifies the interval at which the stacks are collected automatically. For details about the automatic stack collection function determined by this parameter, see "Maintainability > Built-in perf Tool" in *Feature Description*.

Parameter type: integer

Unit: minute

Value range: **0** or 5–60. The value **0** indicates that the automatic stack collection function is disabled. The values 5 to 60 indicate the interval at which the stacks are collected automatically.

NOTE

If you attempt to set this parameter to a value ranging from 1 to 4, this parameter will be automatically adjusted to the default value **5**.

Default value: **5**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

gs_perf_retention_days

Parameter description: Specifies the retention period of flame graph files. For details about the automatic stack collection function determined by this parameter, see "Maintainability > Built-in perf Tool" in *Feature Description*.

Parameter type: integer

Unit: day

Value range: 1 to 8

Default value: **3**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

14.3.29 Security Configuration

enable_security_policy

Parameter description: Specifies whether the unified audit and dynamic data masking policies take effect.

NOTE

Unified audit: The unified audit mechanism is a technology that implements efficient security audit management by customizing audit policies. After the administrator defines the audit object and audit behaviors, if the task executed by a user is associated with an audit policy, the corresponding audit behavior is generated and the audit log is recorded. For details, see "Database Security > Unified Audit" in *Feature Description*.

Dynamic data masking: The dynamic data masking mechanism is a technology that protects privacy data by customizing masking policies. It can effectively prevent unauthorized users from accessing sensitive information while retaining original data. For details, see "Database Security > Dynamic Data Masking" in *Feature Description*.

Parameter type: Boolean

Unit: none

Value range:

on: The security policy is enabled.

off: The security policy is disabled.

Default value: **off**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. Set this parameter to **on** when a security policy is required. However, this occupies system resources and affects system performance.

use_elastic_search

Parameter description: Specifies whether to send unified audit logs to Elasticsearch. If **enable_security_policy** and this parameter are enabled, unified audit logs are sent to Elasticsearch through HTTP or HTTPS (used by default). After this parameter is enabled, ensure that the Elasticsearch service corresponding to **elastic_search_ip_addr** can be properly connected. Otherwise, the process fails to be started.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

on: Unified audit logs are sent to Elasticsearch.

off: Unified audit logs are not sent to Elasticsearch.

Default value: **off**

elastic_search_ip_addr

Parameter description: Specifies the IP address of the Elasticsearch system. If HTTPS is used, the format is **https://***ip*:*port*:*username*. If HTTP is used, the format is **http://***ip*:*port*. In the preceding command, *ip* indicates the IP address of the Elasticsearch server. *port* indicates the listening port for Elasticsearch HTTP communication, and the value ranges from 9200 to 9299. *username* indicates the username used for registering an Elasticsearch account. The initial user is **elastic**. If HTTPS is used, related certificates need to be configured. For details, see "Unified Auditing" in the *Security Hardening Guide*.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string.

Default value: **''**

is_sysadmin

Parameter description: Specifies whether the current user is an initial user.

This is a fixed INTERNAL parameter. It can be viewed but cannot be modified.

Value range: Boolean

on indicates that the user is an initial user.

off indicates that the user is not an initial user.

Default value: **off**

enable_tde

Parameter description: Specifies whether to enable the TDE function. Before creating an encrypted table, set this parameter to **on** and set the **tde_key_info** parameter to configure the key information.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean.

on: The TDE function is enabled.

off: The TDE function is disabled.

Default value: **off**

NOTICE

- If this parameter is set to **on**, ensure that the key information in the **tde_key_info** parameter is correctly configured and the key service can be accessed properly. Otherwise, the database cannot be started properly.
- After this parameter is set to **on** and an encrypted table is created, if this parameter is set to **off** again, data in the existing encrypted table cannot be encrypted or decrypted. As a result, the database is abnormal.

tde_key_info

Parameter description: If TDE is enabled, transparent data needs to access the external key service to implement key management. This parameter is used to configure information about the key service, such as the service address, identity authentication information, and project information.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. For details, see "Security Management > Configuring TDE" in *Administrator Guide*.

Default value: **""**

tde_index_default_encrypt

Parameter description: After this parameter is enabled, if you create an index whose base table is an encrypted table, the database automatically sets the index as an encrypted index, copies encryption parameters such as the encryption algorithms and keys of the base table for the index, and encrypts the data of the index before storing it.

Parameter type: Boolean

Unit: none

Value range:

on: Encryption parameters are automatically set for indexes of encrypted tables.

off: Encryption parameters are not automatically set for indexes in encrypted tables.

Default value: **off**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

block_encryption_mode

Parameter description: Specifies the block encryption mode used by the aes_encrypt and aes_decrypt functions for encryption and decryption.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: enumerated values. Valid values are **aes-128-cbc**, **aes-192-cbc**, **aes-256-cbc**, **aes-128-cfb1**, **aes-192-cfb1**, **aes-256-cfb1**, **aes-128-cfb8**, **aes-192-cfb8**, **aes-256-cfb8**, **aes-128-cfb128**, **aes-192-cfb128**, **aes-256-cfb128**, **aes-128-ofb**, **aes-192-ofb**, and **aes-256-ofb**. **aes** indicates the encryption or decryption algorithm. **128**, **192**, and **256** indicate the key length (unit: bit). **cbc**, **cfb1**, **cfb8**, **cfb128**, and **ofb** indicate the block encryption or decryption mode.

Default value: **aes-128-cbc**

enable_mac_check

Parameter description: Specifies whether label-based mandatory access control takes effect.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean.

on indicates that label-based mandatory access control takes effect.

off indicates that label-based mandatory access control does not take effect.

Default value: **off**

enable_rls_match_index

Parameter description: Specifies whether indexes of a base table can be scanned based on target predicate conditions in row-level security scenarios. Target scenario: The row level security (RLS) policies are set and enabled in the base table, and the query predicate contains the unleakproof system function or like operator.

Value range: Boolean.

on: Base table indexes can be scanned in the target scenario.

off: Base table indexes cannot be scanned in the target scenario.

Default value: **off**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. If hotspot query statements belong to the target scenario and there are few row-level access policies in the base table, enabling this function significantly improves query performance.

NOTE

- The modification of this parameter affects the generation of the execution plan in the target scenario. You can manually invalidate the cache plan by reconnecting to the system or creating operators.
- If this parameter is enabled, the generation of the bitmap scanning operator is affected, and the RLS policy predicate is inserted into the Recheck filter condition. Therefore, when the operator is switched to the lossy mode and there are many RLS policies, the performance is affected.

14.3.30 Global Temporary Table

max_active_global_temporary_table

Parameter description: Specifies whether global temporary tables can be created. The value of this parameter determines the memory reserved in the shared cache for hash tables required by global temporary tables. The total number of active global temporary tables in all sessions is not forcibly limited.

Parameter type: integer

Unit: none

Value range: 0 to 1000000

- **0**: The global temporary table function is disabled.
- > 0: The global temporary table function is enabled.

Default value: **1000**

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

vacuum_gtt_defer_check_age

Parameter description: Checks the differences between the global temporary table relfrozenxid and the ordinary table after VACUUM is executed. WARNING is generated if the difference value exceeds the specified parameter value. Use the default value for this parameter.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 1000000

Default value: **10000**

enable_gtt_concurrent_truncate

Parameter description: Specifies whether to support concurrent execution of TRUNCATE TABLE and DML operations on global temporary tables and concurrent execution of TRUNCATE TABLE on global temporary tables.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**/**true** indicates that the preceding operations can be performed concurrently.
- **off**/**false** indicates that the preceding operations cannot be performed concurrently.

Default value: **on**

14.3.31 HyperLogLog

hll_default_log2m

Parameter description: Specifies the number of buckets for HLL data. The number of buckets affects the precision of distinct values calculated by HLL. The more buckets there are, the smaller the deviation is. The deviation range is as follows: $[-1.04/2^{log2m*1/2}, +1.04/2^{log2m*1/2}]$

Parameter type: integer

Unit: none

Value range: 10 to 16.

Default value: **14**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value or adjust the value based on service requirements.

hll_default_log2explicit

Parameter description: Specifies the default threshold for switching from the explicit mode to the sparse mode.

Parameter type: integer

Unit: none

Value range: 0–12 The value **0** indicates that the explicit mode is skipped. The value 1 to 12 indicates that the mode is switched when the number of distinct values reaches $2^{hll\_default\_log2explicit}$.

Default value: **10**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value or adjust the value based on service requirements.

hll_default_log2sparse

Parameter description: Specifies the default threshold for switching from the **sparse** mode to the **full** mode.

Parameter type: integer

Unit: none

Value range: 0–14 The value **0** indicates that the explicit mode is skipped. The value 1 to 14 indicates that the mode is switched when the number of distinct values reaches $2^{hll\_default\_log2sparse}$.

Default value: **12**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value or adjust the value based on service requirements.

hll_duplicate_check

Parameter description: Specifies whether duplicatecheck is enabled by default.

Parameter type: Boolean

Unit: none

Value range:

- **0**: The function is disabled by default.
- **1**: The function is enabled by default.

Default value: **0**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value or adjust the value based on service requirements.

14.3.32 User-defined Functions

udf_memory_limit

Parameter description: Controls the maximum physical memory that can be used when each database node executes UDFs. This parameter does not take effect in the current version. Use **FencedUDFMemoryLimit** and **UDFWorkerMemHardLimit** to control virtual memory used by fenced udf worker.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer. The value range is from 200 x 1024 to *max_process_memory* and the unit is KB.

Default value: **200MB**

FencedUDFMemoryLimit

Parameter description: Specifies the virtual memory used by each fenced udf worker process.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 KB to 2147483647 KB. The unit can also be MB or GB. **0** indicates that the memory is not limited.

Default value: **0**

UDFWorkerMemHardLimit

Parameter description: Specifies the maximum value of **fencedUDFMemoryLimit**.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 KB to 2147483647 KB. The unit can also be MB or GB.

Default value: **1GB**

14.3.33 Scheduled Task

job_queue_processes

Parameter description: Specifies the number of jobs that can be concurrently executed. This parameter is a POSTMASTER parameter. You can set it using **gs_guc**, and you need to restart **gaussdb** to make the setting take effect.

This parameter is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: 0 to 1000

Function:

- Setting **job_queue_processes** to **0** indicates that the scheduled job function is disabled and that no job will be executed. (Enabling scheduled jobs may affect the system performance. At sites where this function is not required, you are advised to disable it.)
- Setting **job_queue_processes** to a value that is greater than **0** indicates that the scheduled job function is enabled and this value is the maximum number of jobs that can be concurrently processed.

After the scheduled job function is enabled, the job_scheduler thread polls the **pg_job** system catalog at a scheduled interval. The scheduled job check is performed every second by default.

Too many concurrent jobs consume many system resources, so you need to set the number of concurrent jobs to be processed. If the current number of concurrent jobs reaches the value of **job_queue_processes** and some of them expire, these jobs will be postponed to the next polling period. Therefore, you are advised to set the polling interval (the **Interval** parameter of the **submit** interface) based on the execution duration of each job to avoid the problem that jobs in the next polling period cannot be properly processed because of overlong job execution time.

Note: If the number of concurrent jobs is large and the value is too small, these jobs will wait in queues. However, a large parameter value leads to large resource consumption. You are advised to set this parameter to **100** and change it based on the system resource condition.

Default value: **10**

enable_prevent_job_task_startup

Parameter description: Specifies whether to start the job thread.

Value range: Boolean

Unit: none

- **on** indicates that the job thread is not started.
- **off** indicates that the job thread is started.

Default value: **off**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

14.3.34 Thread Pool

enable_thread_pool

Parameter description: Specifies whether to enable the thread pool function. This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**. You are advised to disable this parameter in performance-sensitive scenarios with low-concurrency and persistent connection and enable this parameter in other scenarios.

Value range: Boolean

- **on** indicates that the thread pool function is enabled.
- **off** indicates that the thread pool function is disabled.

Default value: **on**

thread_pool_attr

Parameter description: Specifies the detailed attributes of the thread pool function. This parameter is valid only when **enable_thread_pool** is set to **on**. Only the sysadmin user can access this parameter.

Parameter type: string

Unit: none

Value range:

This parameter consists of three parts: **'thread_num'**, **'group_num'**, and **'cpubind_info'**. The meanings of the three parts are as follows:

- **thread_num** is the total number of initial threads in the thread pool, which can be dynamically expanded. The value ranges from 0 to 4096. The value **0** indicates that the database automatically configures the number of threads in the thread pool based on the number of CPU cores. If the value is greater than **0**, the number of threads in the thread pool is the same as the value of **thread_num**. You are advised to set the thread pool size based on the

hardware configuration. The formula is as follows: Value of **thread_num** = Number of CPU cores x 3–5. The maximum value of **thread_num** is **4096**.

- **group_num** indicates the number of thread groups in the thread pool. The value ranges from 0 to 64. The value **0** indicates that the database automatically configures the number of thread groups in the thread pool based on the number of NUMA groups. If the value is greater than **0**, the number of thread groups in the thread pool is the same as the value of **group_num**.
- **cpubind_info** specifies whether the thread pool is bound to a core. The available configuration modes are as follows: 1. **'(nobind)'**: The thread is not bound to a core. 2. **'(allbind)'**: Use all CPU cores that can be queried in the current system to bind threads. 3. **'(nodebind: 1, 2)'**: Use the CPU cores in NUMA groups 1 and 2 to bind threads. 4. **'(cpubind: 0-30)'**: Use CPU cores 0 to 30 to bind threads. 5. **'(numabind: 0-30)'**: Use CPU cores 0 to 30 in the NUMA group to bind threads. This parameter is case-insensitive. This parameter does not take effect in the resource multi-tenancy mode.

Default value: **'4096,2,(nobind)'** (196-core CPU/1536 GB memory, 128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory); **'2048,2,(nobind)'** (96-core CPU/768 GB memory, 80-core CPU/640 GB memory); **'1024,2,(nobind)'** (64-core CPU/512 GB memory, 60-core CPU/480 GB memory, 32-core CPU/256 GB memory); **'512,2,(nobind)'** (16-core CPU/128 GB memory); **'256,2,(nobind)'** (8-core CPU/64 GB memory); **'128,2,(nobind)'** (4-core CPU/32 GB memory); **'32,1,(nobind)'** (4-core CPU/16 GB memory)

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: When the memory is sufficient and the CPU performance is good, increase the value of this parameter if the service requires more connections.

thread_pool_stream_attr

Parameter description: Specifies the detailed attributes of the stream thread pool function. This parameter is valid only when **enable_thread_pool** is set to **on** and only takes effect on DNs. Only the sysadmin user can access this parameter. This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string, consisting of one or more characters

This parameter consists of four parts: **'stream_thread_num'**, **'stream_proc_ratio'**, **'group_num'**, and **'cpubind_info'**. The meanings of the four parts are as follows:

- **stream_thread_num** indicates the total number of threads in the stream thread pool. The value ranges from 0 to 4096. The value **0** indicates that the database automatically configures the number of threads in the thread pool based on the number of CPU cores. If the value is greater than **0**, the number of threads in the thread pool is the same as the value of **stream_thread_num**. You are advised to set the thread pool size based on the hardware configuration. The formula is as follows: Value of **stream_thread_num** = Number of CPU cores x 3–5. The maximum value of **stream_thread_num** is **4096**.
- **stream_proc_ratio** indicates the ratio of proc resources reserved for stream threads. The value is a floating-point number. The default value is **0.2**. The

reserved proc resources are calculated as follows: **stream_proc_ratio** x **stream_thread_num**.

- **group_num** indicates the number of thread groups in the thread pool. The value ranges from 0 to 64. The value **0** indicates that the database automatically configures the number of thread groups in the thread pool based on the number of NUMA groups. If the value is greater than **0**, the number of thread groups in the thread pool is the same as the value of **group_num**. The value of **group_num** in **thread_pool_stream_attr** must be the same as that in **thread_pool_attr**. If they are set to different values, the value of **group_num** in **thread_pool_attr** is used.
- **cpubind_info** specifies whether the thread pool is bound to a core. The available configuration modes are as follows: 1. **'(nobind)'**: The thread is not bound to a core. 2. **'(allbind)'**: Use all CPU cores that can be queried in the current system to bind threads. 3. **'(nodebind: 1, 2)'**: Use the CPU cores in NUMA groups 1 and 2 to bind threads. 4. **'(cpubind: 0-30)'**: Use CPU cores 0 to 30 to bind threads. 5. **'(numabind: 0-30)'**: Use CPU cores 0 to 30 in the NUMA group to bind threads. This parameter is case-insensitive. The value of **cpubind_info** in **thread_pool_stream_attr** must be the same as that in **thread_pool_attr**. If they are set to different values, the value of **cpubind_info** in **thread_pool_attr** is used.

Default value:

stream_thread_num: **16**

stream_proc_ratio: **0.2**

group_num and **cpubind_info**: For details, see **thread_pool_attr**.

resilience_threadpool_reject_cond

Parameter description: Specifies the percentage of thread pool usage for escape from overload. This parameter takes effect only when the GUC parameters **use_workload_manager** and **enable_thread_pool** are enabled. This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string, consisting of one or more characters

This parameter consists of **recover_threadpool_percent** and **overload_threadpool_percent**. The meanings of the two parts are as follows:

- **recover_threadpool_percent**: specifies the thread pool usage when the thread pool recovers to the normal state. When the thread pool usage is less than the value of this parameter, the escape from overload function is disabled and new connections are allowed. The value ranges from 0 to *INT_MAX*. The value indicates a percentage.
- **overload_threadpool_percent**: specifies thread pool usage when the thread pool is overloaded. If the thread pool usage is greater than the value of this parameter, the current thread pool is overloaded. In this case, the escape from overload function is enabled to kill sessions and forbid new connections. The value ranges from 0 to *INT_MAX*. The value indicates a percentage.

Default value: **'0,0'**, indicating that the thread pool escape function is disabled.

Example:


```
resilience_threadpool_reject_cond = '50,90'
```

When the thread pool usage exceeds 90%, new connections are forbidden and stacked sessions are killed. When the thread pool usage decreases to 50%, session killing is stopped and new connections are allowed.

NOTICE

- The thread pool usage can be queried in the DBE_PERF.local_threadpool_status view. The initial number of threads in the thread pool can be obtained by querying the **thread_pool_attr** parameter.
- If this parameter is set to a small value, the thread pool escape from overload process is frequently triggered. As a result, ongoing sessions are forcibly logged out, and new connections fail to be connected for a short period of time. Therefore, exercise caution when setting this parameter based on the actual thread pool usage.
- The values of **recover_threadpool_percent** and **overload_threadpool_percent** can be **0** at the same time. In addition, the value of **recover_threadpool_percent** must be smaller than that of **overload_threadpool_percent**. Otherwise, the setting does not take effect.

14.3.35 Backup and Restoration

operation_mode

Parameter description: Specifies whether the system enters the backup and restoration mode.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the system is in the backup and restoration mode.
- **off** indicates that the system is not in the backup and restoration mode.

Default value: **off**

enable_cbm_tracking

Parameter description: This parameter must be enabled when Roach is used to perform full and incremental backups. If this parameter is disabled, the backup will fail.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: The cbm tracking is enabled.
- **off**: The cbm tracking is disabled.

Default value: **off**

max_size_for_xlog_retention

Parameter description: Specifies when to forcibly update the backup replication slot or logical replication slot to prevent the disk from being full and the instance from being read-only because logs cannot be recycled during backup or logical decoding. It is recommended that the value of this parameter be a little smaller than the value of **datastorage_threshold_value_check** of the CM Server component to prevent the instance from entering the read-only state.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: –100 to 2147483647

- The value **0** indicates that this function is disabled.
- A negative value indicates that the backup replication slot or logic replication slot is forcibly updated when the disk usage exceeds the threshold and logs are recycled due to blocked backup operation or logical decoding. For example, **–80** indicates that the disk usage exceeds 80% of the threshold.
- A positive value indicates that the backup replication slot or logic replication slot is forcibly updated when the size of stacked logs exceeds the threshold and logs are recycled due to blocked backup operation or logical decoding. For example, **32** indicates that the backup replication slot lags behind the redo location of the current checkpoint by more than 32 log segments. (The size of each log segment is 16 MB.)

Default value: **–80**

max_cbm_retention_time

Parameter description: Specifies the interval at which CBM backup files are forcibly recycled. If CBM files cannot be recycled during backup, the disk will be full and the instance will be read-only. You are advised to set this parameter based on the full backup interval.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: 86400 to 2147483647

- The unit is second.
- The minimum value is 1 day.
- The default value is 2 weeks.

Default value: 1209600

14.3.36 Undo

undo_space_limit_size

Parameter description: Specifies the threshold for forcibly recycling undo space. When the undo space usage reaches 80% of the threshold, forcible recycling starts. You are advised to set **undo_space_limit_size** to a value greater than or equal to that of **undo_limit_size_per_transaction**.

Parameter type: integer

Unit: page (8 KB)

Value range: 102400 to 2147483647. The default unit is page.

Default value: **256GB**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**. For example, the value **10000** indicates 10000 pages, and the value **10000KB** indicates 10000 KB. If the value contains a unit, the value can be KB, MB, or GB, but cannot be TB.

Setting suggestion: Retain the default value. If the disk space is too small to meet the default value requirements, set this parameter to a smaller value. If long transactions or large transactions exist in your service, set **undo_space_limit_size** to a larger value based on the disk usage. For details about the value setting, see the **info** column of the **gs_stat_undo** system function in "SQL Reference > Functions and Operators > System Administration Functions > Undo System Functions > gs_stat_undo parameters" in *Developer Guide*.

undo_limit_size_per_transaction

Parameter description: Specifies the undo space threshold of a single transaction. If the threshold is reached, the transaction is rolled back due to an error. You are advised to set **undo_limit_size_per_transaction** to a value less than or equal to that of **undo_space_limit_size**. If the value of **undo_limit_size_per_transaction** is greater than that of **undo_space_limit_size**, the displayed value is the same as the configured value when you run the **show undo_limit_size_per_transaction** command to query the parameter value. The only difference is that the smaller value between **undo_space_limit_size** and **undo_limit_size_per_transaction** is used as the actual undo space threshold of a single transaction.

Parameter type: integer

Unit: page (8 KB)

Value range: 256 to 2147483647. The default unit is page. If this parameter is set to a value greater than 134217728 (that is, 1 TB), the value **134217728** takes effect.

Default value: **32GB**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**. For example, if the value is **10000**, the **undo_limit_size_per_transaction** is 10000 pages. If the value is **10000 KB**, the **undo_limit_size_per_transaction** is 10000 KB. If the value contains a unit, the value can be KB, MB, or GB, but cannot be TB.

Setting suggestion: Retain the default value. If the disk or memory space is too small to meet the requirements of the default value, set this parameter to a smaller value. If long transactions or large transactions exist in your service, and the number of Undo records generated in a single transaction is greater than the value of this parameter, set **undo_limit_size_per_transaction** to a larger value based on the disk usage. For details about the value setting, see the info column of the **gs_stat_undo** system function in "SQL Reference > Functions and Operators > System Administration Functions > Undo System Functions > gs_stat_undo parameters" in *Developer Guide*. If **undo_limit_size_per_transaction** is set to a value greater than 1 TB, the system performance and stability may be affected. Therefore, if the parameter is set to a value greater than 134217728 (that is, 1 TB), the configured value is displayed when you run the **show**

undo_limit_size_per_transaction command to query the parameter value, but the value **134217728** takes effect.

14.3.37 DCF Parameters Settings

enable_dcf

Parameter description: Specifies whether to enable the DCF mode.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean. The value can be **on** or **off**. **on** indicates that the current log replication mode is DCF, and **off** indicates that the current log replication mode is not DCF.

Default value: **off**

dcf_ssl

Parameter description: This parameter is no longer used. The DCF reuses the GUC parameter **ssl**. For details, see section "Security and Authentication."

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean. The value can be **on** or **off**. The value **on** indicates that SSL is used, and the value **off** indicates that SSL is not used.

Default value: **on**

dcf_config

Parameter description: Specifies the DCF cluster configuration information, which is configured by the OM during installation and cannot be modified after installation.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Default value: a character string, which is configured by the OM during installation.

dcf_data_path

Parameter description: Specifies the DCF data path, which is configured by the OM during installation and cannot be modified after installation.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Default value: a string, which is the **dcf_data** directory under the data directory of the DN.

dcf_log_path

Parameter description: Specifies the DCF log path, which is configured by the OM during installation and cannot be modified after installation.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Default value: a string, which is the **dcf_log** directory under the data directory of the DN.

dcf_node_id

Parameter description: Specifies the ID of the DN where the DCF is located. This parameter is defined by the user during installation and mode switching.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Default value: an integer, which is specified by users during installation.

dcf_max_workers

Parameter description: Specifies the largest number of DCF callback function threads. The DCF needs to apply for the shared memory and semaphore before it invokes the function registered by the DN with the DCF through the callback function.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 262143.

Default value: **40**

dcf_truncate_threshold

Parameter description: Specifies the threshold for a DN to truncate DCF logs.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to 2147483647.

Default value: **100000**

dcf_election_timeout

Parameter description: Specifies the timeout interval for selecting the DCF leader and follower. The election timeout interval depends on the status of the network between DNs. If the timeout interval is short and the network quality is poor, timeout occurs. After the network recovers, the election becomes normal. You are advised to set a proper timeout interval based on the current network status. Restriction on the DCF node clock: The maximum clock difference between DCF nodes is less than half of the election timeout interval. In DCF manual election mode, to ensure timely CM arbitration, do not modify this parameter. Instead, use the default election timeout interval.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to 600, in seconds.

Default value: **3**

dcf_enable_auto_election_priority

Parameter description: Specifies whether the DCF priority can be automatically adjusted. The value **0** indicates that automatic adjustment is not allowed, and the value **1** indicates that automatic adjustment is allowed.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer, **0** or **1**.

Default value: **1**

dcf_election_switch_threshold

Parameter description: Specifies the DCF threshold for preventing frequent switchover to primary. It is recommended that this parameter be set based on the maximum fault duration acceptable for user services.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 2147483647, in seconds.

Default value: **0**

dcf_run_mode

Parameter description: Specifies the DCF election mode. The value **0** indicates the automatic election mode, the value **1** indicates the manual election mode, and the value **2** indicates that the election mode is disabled. Currently, the election mode can be disabled only in minority restoration scenarios. If the election mode is disabled, the database instance will become unavailable.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Note: The working mode of an instance can be switched only when the instance is running properly. Otherwise, the instance is still abnormal after the switching. The DCF working mode configured in GUC parameters must be the same as that configured by using cm_ctl. That is, both DCF working modes must be set to manual or automatic at the same time.

For example, to set the DCF manual mode, run the following command:

```
cm_ctl set --param --server -k dn_arbitrate_mode=quorum
cm_ctl reload --param --server
gs_guc reload -Z datanode -I all -N all  -c "dcf_run_mode=1"
```

To set the DCF automatic mode, run the following command:

```
cm_ctl set --param --server -k dn_arbitrate_mode=paxos
cm_ctl reload --param --server
gs_guc reload -Z datanode -I all -N all  -c "dcf_run_mode=0"
```

Value range: **0**, **1**, or **2**

Default value: **1**

dcf_log_level

Parameter description: Specifies the DCF log level.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string

- To disable the log function, set this parameter to **"NONE"**, which cannot be used together with the following log levels:
- To enable the log function, set this parameter to one or a combination of the following log levels: **"RUN_ERR|RUN_WAR|RUN_INF|DEBUG_ERR|DEBUG_WAR|DEBUG_INF|TRACE|PROFILE|OPER"**.

  If two or more log levels are used together, separate them with vertical bars (|). The log level cannot be set to an empty string.

Default value: **"RUN_ERR|RUN_WAR|DEBUG_ERR|OPER|RUN_INF|PROFILE"**

dcf_log_backup_file_count

Parameter description: Specifies the number of DCF run log backups.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to 1000.

Default value: **100**

dcf_max_log_file_size

Parameter description: Specifies the maximum size of a DCF run log file.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to 1000, in MB.

Default value: **10**

dcf_socket_timeout

Parameter description: Specifies the timeout interval for the DCF communication module to connect to the socket. This parameter takes effect upon the system restart. In an environment where the network quality is poor, if the timeout interval is set to a small value, a connection may fail to be set up. In this case, you need to increase the value.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 10 to 600000, in ms.

Default value: **5000**

dcf_connect_timeout

Parameter description: Specifies the timeout interval for the DCF communication module to set up a connection. This parameter takes effect upon the system restart. In an environment where the network quality is poor, if the timeout interval is set to a small value, a connection may fail to be set up. In this case, you need to increase the value.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 10 to 600000, in ms.

Default value: 60000

dcf_mec_fragment_size

Parameter description: Specifies the fragment size of the DCF communication module. This parameter takes effect upon the system restart.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 32 to 10240, in KB.

Default value: **64**

dcf_stg_pool_max_size

Parameter description: Specifies the maximum size of the memory pool of the DCF storage module. This parameter takes effect upon the system restart.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 32 to 2147483647, in MB.

Default value: **2048**

dcf_stg_pool_init_size

Parameter description: Specifies the minimum size of the memory pool of the DCF storage module. This parameter takes effect upon the system restart.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 32 to 2147483647, in MB.

Default value: **32**

dcf_mec_pool_max_size

Parameter description: Specifies the maximum size of the memory pool of the DCF communication module. This parameter takes effect upon the system restart.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 32 to 2147483647, in MB.

Default value: **200**

dcf_flow_control_disk_rawait_threshold

Parameter description: Specifies the disk waiting threshold for DCF flow control.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 2147483647, in μs.

Default value: **100000**

dcf_flow_control_net_queue_message_num_threshold

Parameter description: Specifies the threshold for the number of messages in a network queue for DCF flow control.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 2147483647.

Default value: 1024

dcf_flow_control_cpu_threshold

Parameter description: Specifies the threshold for DCF CPU flow control.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 2147483647, in percentage (%).

Default value: **100**

dcf_mec_batch_size

Parameter description: Specifies the number of batch messages for DCF communication. When the value is **0**, the DCF automatically adjusts the value based on the network and the amount of data to be written. This parameter takes effect upon the system restart.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 1024.

Default value: **0**

dcf_mem_pool_max_size

Parameter description: Specifies the maximum DCF memory. This parameter takes effect upon the system restart.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 32 to 2147483647, in MB.

Default value: **2048**

dcf_mem_pool_init_size

Parameter description: Specifies the initial size of the DCF memory. This parameter takes effect upon the system restart.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 32 to 2147483647, in MB.

Default value: **32**

dcf_compress_algorithm

Parameter description: Specifies the compression algorithm for DCF run log transmission.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer

- The value **0** indicates no compression.
- **1** indicates the LZ4 compression algorithm.

Default value: **0**

dcf_compress_level

Parameter description: Specifies the compression level for DCF log transmission. Before this parameter takes effect, a valid compression algorithm must be configured, that is, the **dcf_compress_algorithm** parameter is set.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to 22.

If compression is disabled, the configured compression level does not take effect.

Default value: **1**

dcf_mec_channel_num

Parameter description: Specifies the number of DCF communication channels. This parameter takes effect upon the system restart.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to 64.

Default value: **1**

dcf_rep_append_thread_num

Parameter description: Specifies the number of DCF log replication threads. This parameter takes effect upon the system restart.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to 1000.

Default value: **2**

dcf_mec_agent_thread_num

Parameter description: Specifies the number of DCF communication working threads. This parameter takes effect upon the system restart. It is recommended

that the value of **dcf_mec_agent_thread_num** be greater than or equal to 2 x Number of nodes x Value of **dcf_mec_channel_num**.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to 1000.

Default value: **10**

dcf_mec_reactor_thread_num

Parameter description: Specifies the number of reactor threads used by the DCF. This parameter takes effect upon the system restart. It is recommended that the ratio of the value of **dcf_mec_reactor_thread_num** to the value of **dcf_mec_agent_thread_num** be 1:40.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to 100.

Default value: **1**

dcf_log_file_permission

Parameter description: Specifies the attribute of the DCF run log file. The parameter setting takes effect after the system is restarted. This parameter is configured during installation and cannot be modified. To allow other users in the same group to access logs, ensure that all parent directories can be accessed by other users in the same group. That is, if **dcf_log_path_permission** is set to **750**, **dcf_log_file_permission** can only be set to **600** or **640**. If **dcf_log_path_permission** is set to **700**, **dcf_log_file_permission** must be set to **600**.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: enumerated type. The value can be **600** or **640**.

Default value: **600**

dcf_log_path_permission

Parameter description: Specifies the attribute of the DCF run log directory. The parameter setting takes effect after the system is restarted. This parameter is configured during installation and cannot be modified. To allow other users in the same group to access the log path, set this parameter to **750**. Otherwise, set this parameter to **700**.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: enumerated type. The value can be **700** or **750**.

Default value: **700**

dcf_majority_groups

Parameter description: Sets the DCF policy-based majority function. For a group that requires this parameter, at least one standby node in the group receives logs. That is, there is a synchronous standby node in the group. If nodes are added to or deleted from the DCF instance or the group value of a node in the instance is changed, you need to modify the configuration accordingly. When modifying this parameter, ensure that the value of **group** exists and is valid.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string

- To disable the policy-based majority function, enter an empty string **""**.
- To enable the function, configure valid group values separated by commas (,). The group values must exist in **dcf_config**. For example, if the group values 1 and 2 are added to the DCF policy-based majority configuration, you can set this parameter to **"1,2"**. If the group value does not exist in **dcf_config** or other characters are configured, the DCF considers the configured group invalid.

Default value: an empty string

CAUTION

If all nodes in a group are faulty after the parameter is configured, you need to remove the group from the parameter list when performing node build operations (node recovery or node replacement without changing the IP address) on a node. After the node recovers, you can configure the group again.

dcf_node_id_map

Parameter description: Specifies the dictionary mapping between standby DN names and DCF node IDs. The parameter setting takes effect after the system is restarted. This parameter is configured during installation and cannot be modified. This parameter is used in DCF cluster installation, upgrade, and node replacement scenarios. The value of **standby_name** configured in the GUC parameter **synchronous_standby_names** must be included in this dictionary.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string. The configuration format is 'standby_name1:dcf_node_id1,standby_name2:dcf_node_id2'. The values of standby DN names and the corresponding DCF node IDs are separated by commas (,).

Default value: an empty string

dcf_candidate_names

Parameter description: Specifies the DCF candidate list. That is, names of DNs that may be selected as the primary node. In DCF automatic mode, the election policy is controlled by this parameter. DNs not in the list cannot be elected as a primary node.

Parameter type: string

Unit: none

Value range: a string in the format of 'dn_name1,dn_name2,dn_name3'. The parameter depends on **dcf_node_id_map**. DN names must be in **dcf_node_id_map** and separated by commas (,).

Default value: an empty string

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none

dcf_thread_effective_time

Parameter description: Specifies the effective time of the DCF flushing thread. This parameter is used to determine whether the disk I/O hangs. If the DCF control log cannot update and I/O resources cannot be accessed within the period specified by this parameter, the DCF considers that the thread I/O hangs and the switchover is triggered. If this parameter is set to **0**, the I/O hang detection is disabled.

Parameter type: integer

Unit: second

Value range: 0 to 1000

Default value: 160

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none

dcf_pri_leader_timeout

Parameter description: Specifies the timeout interval for priority-based leader election. In DCF automatic mode, after priority-based leader election is enabled, the backup node triggers priority-based leader election. If the backup node is not elected as the leader within the timeout interval, the election is canceled. If this parameter is set to **0**, the priority-based selection of the primary node waits until the election is successful.

Parameter type: integer

Unit: second

Value range: 0 to 3600

Default value: **60**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none

dcf_static_leader_timeout

Parameter description: Specifies the timeout interval for preferentially electing the old leader. In DCF automatic mode, after the cluster is restarted, the old leader is preferentially elected. If the election fails within the timeout interval, the election is canceled.

Parameter type: integer

Unit: second

Value range: 0 to 600

Default value: **60**

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: none

14.3.38 Flashback

This section describes parameters related to the flashback function. In this version, only the Ustore engine supports flashback, while the Astore engine does not support flashback.

enable_recyclebin

Parameter description: Specifies whether the recycle bin is enabled or disabled in real time.

Parameter type: Boolean

Value range:

- **on** indicates that the recycle bin is enabled in real time.
- **off** indicates that the recycle bin is disabled in real time.

Default value: **off**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: To use the flashback table function, set **enable_recyclebin** to **on**.

recyclebin_retention_time

Parameter description: Specifies the retention period of objects in the recycle bin. The objects will be automatically deleted after the retention period expires.

Parameter type: integer

Unit: s

Value range: 1 to 2147483647.

Default value: **15 min** (900s)

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**. For example, if the value is **900** without a unit, **recyclebin_retention_time** indicates 900s. If the value is **15min**, **recyclebin_retention_time** indicates 15 minutes. If the unit is required, the value must be **s**, **min**, **h**, or **d**.

version_retention_age

Parameter description: Specifies the number of transactions retained in earlier versions. The earlier versions with the number of transactions larger than the value of this parameter will be recycled and cleared.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 576460752303423487. **0** means no delay.

Default value: **0**

CAUTION

This parameter has been deprecated.

vacuum_defer_cleanup_age

Parameter description: Determines whether a dead tuple can be physically deleted based on the global minimum active transaction **oldestxmin**. If the value is less than **oldestxmin**, the dead tuple is physically deleted. If **vacuum_defer_cleanup_age** is set to a value greater than 0, the recycling of dead tuples needs to be delayed. In this case, the result of **oldestxmin** minus **vacuum_defer_cleanup_age** is used as the new threshold for clearing dead tuples.

For example, if **oldestxmin** and **vacuum_defer_cleanup_age** are set to **200** and **50**, respectively, the new threshold is 150 (that is, 200 – 50). The tuples whose XID < 150 will be physically deleted.

Parameter type: integer

Unit: none

Value range: 0 to 1000000. **0** means no delay.

Default value: **0**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

undo_retention_time

Parameter description: Specifies the period for retaining undo logs of earlier versions.

Parameter type: integer

Unit: s

Value range: 0 to 259200.

Default value: **0**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

CAUTION

1. If this parameter is set to **0** during the Ustore flashback query, the snapshot information at the flashback point will be cleared. In earlier versions, no flashback query can be performed. When a flashback query is performed, the error message "cannot find the restore point" is displayed.
2. If the time within which the undo logs of earlier versions need to be retained is time1 and the SQL statement execution time for the flashback query is time2, you need to set **undo_retention_time** to a value greater than time1 + time2. That is, set **undo_retention_time** to a value greater than time1 + time2 + 3s. You are advised to set **undo_retention_time** to a value equal to time1 + 1.5 x time2. For example, if you want to retain the logs of earlier versions within the latest 3 hours, and the SQL statement execution time for the flashback query is 1 hour, set **undo_retention_time** to a value equal to 3 hours + 1.5 x 1 hour, that is, 4.5 hours.

14.3.39 Rollback Parameters

max_undo_workers

Parameter description: Specifies the number of undo worker threads invoked during asynchronous rollback. The parameter setting takes effect after the system is restarted.

This parameter is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to 100

Default value: **5**

14.3.40 AI Features

enable_hypo_index

Parameter description: Specifies whether the database optimizer considers the created virtual index when executing **EXPLAIN**. By executing **EXPLAIN** on a specific query statement, you can evaluate whether the index can improve the execution efficiency of the query statement based on the execution plan provided by the optimizer.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: A virtual index is created when the **EXPLAIN** command is executed.
- **off**: No virtual index is created when the **EXPLAIN** command is executed.

Default value: **off**

db4ai_snapshot_mode

Parameter description: There are two snapshot modes: MSS (materialized mode, storing data entities) and CSS (computing mode, storing incremental information).

Parameter type: string

Unit: none

Value range: **"MSS"** or **"CSS"**

- **MSS**: materialized mode. The DB4AI stores data entities when snapshots are created.
- **CSS**: computing mode. The DB4AI stores incremental information when snapshots are created.

Default value: "MSS"

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. Switch the mode as required.

db4ai_snapshot_version_delimiter

Parameter description: Specifies the delimiter for the snapshot version of a data table.

Parameter type: string

Unit: none

Value range: a string of one character, such as **@** and **#**. Note that **!** is invalid.

Default value: **"@"**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

db4ai_snapshot_version_separator

Parameter description: Specifies the delimiter for the snapshot subversion of a data table.

Parameter type: string

Unit: none

Value range: a string of one character, such as **@** and **#**. Note that **?** is invalid.

Default value: **"."**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

enable_ai_stats

Parameter description: Specifies whether to create or use intelligent statistics.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that intelligent statistics are created and used.
- **off** indicates that intelligent statistics are not created or used.

Default value: **on**

multi_stats_type

Parameter description: Specifies the type of statistics to be created when **enable_ai_stats** is set to **on**.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: enumerated values. Valid values include **"BAYESNET"**, **"MCV"**, and **"ALL"**.

- **"BAYESNET"**: Only intelligent statistics are created.
- **"MCV"**: Only traditional statistics are created.
- **"ALL"**: Both traditional statistics and intelligent statistics are created.

Default value: **"BAYESNET"**

ai_stats_cache_limit

Parameter description: Specifies the maximum number of models that can be cached when **enable_ai_stats** is set to **on**.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 30 to 1000

Default value: **100**

enable_operator_prefer

Parameter description: Specifies whether to enable the operator preference rule. If the estimated costs are similar, the parameterized path is preferred for table join.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that parameterized path preference is enabled.
- **off** indicates that parameterized path preference is disabled.

Default value: **off**

CAUTION

There are two prerequisites for this parameter to take effect:

- The parameterized path is generated.
- The estimated cost of the parameterized path is similar to that of other index scan operators.

enable_cachedplan_mgr

Parameter description: Specifies whether to enable the adaptive plan selection function. Adaptive plan selection solves the performance issue caused by the traditional single cache plan that cannot change according to the query condition parameter, and avoids frequent calling of query optimizers. Users can enable this function and maintain multiple cache plans for adapting to different query parameters, improving query execution performance.

Parameter type: Boolean.

Unit: none

Value range:

- **on** indicates that the multi-plan cache function is allowed.
- **off** indicates that the multi-plan cache function is not allowed.

Default value: **on**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. Enabling this parameter helps improve the query performance. If this parameter is disabled, the query performance may deteriorate.

max_stmt_aplan_num

Parameter description: Specifies the maximum number of candidate plans per query in adaptive plan selection.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 20

Default value: **5**

recommend_session_aplan_memory

Parameter description: Specifies the maximum memory size of candidate plans of each session in adaptive plan selection. If the size is greater than or equal to the value of this parameter, no new candidate plan will be inserted into the memory. The unit is KB.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1024 to 102400

Default value: **5120**

repick_plan_min_duration

Parameter description: Specifies the lower limit of the detected plan. During policy detection, if the execution time of the detected policy is not less than that of the cplan multiplied by the value of **repick_plan_min_duration**, an error is reported. If this parameter is set to **0**, the error reporting mechanism is disabled.

This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to *INT_MAX*

Default value: **0**

enable_adaptive_cost

Parameter description: Determines whether to enable the feedback-based optimizer cardinality and cost correction functions. If this parameter is enabled, the processes of collecting operator information and estimating cardinality are enabled, and the thread for starting backend model maintenance is enabled; otherwise, the thread exits.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: The operator information and cardinality estimation processes are enabled.
- **off**: The operator information and cardinality estimation processes are disabled.

Default value:

- **on**: default value of the newly installed database.
- **off**: default value of the database in versions earlier than 505.1.0 after the database is upgraded.

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. Enable or disable this parameter based on the actual requirement.

enable_feedback_cardest

Parameter description: Specifies whether to enable the feedback-based optimizer cardinality and cost correction functions. This parameter is used by developers to diagnose model-related problems. Typically **enable_adaptive_cost** is set to **off**. If it is set to **on**, the operator information is still collected and the API of cardinality estimation feedback is still called. However, in this case, the thread for backend automatic model maintenance is not enabled. Developers can use the

gs_acm_analyze_workload_manual() function to manually train models for diagnosing problems.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: The operator information is collected, and the API of cardinality estimation feedback is still called. However, the backend automatic model maintenance thread is not enabled.
- **off**: The **enable_adaptive_cost** parameter takes full control.

Default value:

- **on**: default value of the newly installed database.
- **off**: default value of the database in versions earlier than 505.1.0 after the database is upgraded.

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: You are advised to retain the default value so that developers can diagnose model-related problems. If this parameter is not required, you can disable it and the **enable_adaptive_cost** parameter can take full control. However, this operation will compromise flexibility.

adaptive_cardest_strategy

Parameter description: Specifies the preference of the selection cardinality estimation model.

Value type: enumerated type

Unit: none

Value range: **"auto"**, **"use_statistics"**, and **"use_feedback"**.

- **"auto"**: adaptive mode. In this mode, the system automatically determines whether to use the statistical method or feedback method based on the estimation accuracy in the history.
- **"use_statistics"**: Statistics are preferentially used for cardinality estimation.
- **"use_feedback"**: The feedback model is preferentially used for cardinality estimation.

Default value: **"auto"**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

maximal_feedback_model_num

Parameter description: Specifies the maximum number of cardinality feedback models. If the actual number exceeds the value of this parameter, no new models will be trained.

Parameter type: integer.

Unit: none

Value range: –1 to 1000000. The value **–1** indicates that there is no upper limit.

Default value: **10000**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. If this parameter is set to an excessively small value, the cardinality estimation feedback function may fail because the system cannot train enough new models. If this parameter is set to an excessively large value, the system performance may deteriorate.

feedback_model_cache_limit

Parameter description: Specifies the maximum number of cardinality feedback models that can be cached in the global memory.

Parameter type: integer.

Unit: none

Value range: 10 to 100000

Default value: **500**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. If this parameter is set to an excessively large or small value, the system performance deteriorates.

feedback_model_expired_time

Parameter description: Specifies the timeout interval for cardinality feedback models, which is used to clear expired models periodically.

Parameter type: integer.

Unit: millisecond

Value range: 500–2147483647

Default value: **1000*60*60*24**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. If the timeout interval is too long, the system performance deteriorates. If the timeout interval is too short, unexpired models may be deleted, affecting functions.

feedback_collection_expired_time

Parameter description: Specifies the timeout interval for cardinality feedback models, which is used to clear expired operator models periodically.

Parameter type: integer.

Unit: millisecond

Value range: 50–2147483647

Default value: **1000*60*60**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. If the timeout interval is too long, the system performance deteriorates. If the timeout interval is too short, unexpired models may be deleted, affecting functions.

adaptive_cost_min_time

Parameter description: Specifies the execution duration threshold of SQL statements for cardinality feedback collection. Only the feedback of statements whose execution duration is greater than the value of this parameter is collected.

Parameter type: integer.

Unit: millisecond

Value range: 0 to 2147483647

Default value: 1000

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: You are advised to set this parameter to 20% of the execution duration of slow queries to be tuned. If this parameter is set to a small value, the system performance may deteriorate by about 1%. If this parameter is set to a large value, the query that can be automatically optimized does not take effect.

cost_update_window_size

Parameter description: Specifies the size of the sliding window for collecting data for regression.

Parameter type: integer.

Unit: none

Value range: 1–20

Default value: **5**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

adaptive_costest_strategy

Parameter description: Specifies whether the new or old cost model is used for cost evaluation.

Value type: enumerated type

Unit: none

Value range: **"L0"** and **"L1"**

- **"L0"**: The new cost model is triggered only when the cardinality estimation is correct (for example, using the cardinality estimation feedback).
- **"L1"**: The new cost model is preferentially used for calculation.

Default value: **"L0"**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value.

adaptive_costmodel_calibration_interval

Parameter description: Specifies the interval at which the cost model correction logic is triggered.

Parameter type: integer.

Unit: millisecond

Value range: 0 to 2147483647. The value **0** indicates that the cost models are not automatically corrected.

Default value: **1000*60*60**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. If this parameter is set to an excessively small value, the cost model frequently triggers the correction function, resulting in performance deterioration. If this parameter is set to an excessively large value, the model cannot be corrected in a timely manner, resulting in poor performance.

unix_socket_directory

Parameter description: Specifies the path for storing files in the unix_socket communication mode. You can set this parameter only in the configuration file **gaussdb.conf**. Before enabling the fenced mode, you need to set this GUC parameter.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a string of 0 or more characters

Default value: ''

enable_ai_watchdog

Parameter description: Enables or disables the AI watchdog function.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **on**

enable_ai_watchdog_forcible_oom_detection

Parameter description: Forcibly enables or disables the OOM detection function of the AI watchdog. If this parameter is disabled, the system automatically determines whether to enable the OOM detection function based on the current database specifications. In automatic determination mode, the OOM detection function is enabled only when **max_process_memory** is set to **64GB** or a larger value. The OOM detection function depends on the information obtained by the memory management module. Therefore, if the memory management module is disabled or invalid, the OOM detection function cannot be enabled.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: enabled.
- **off**: disabled.

Default value: **off**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value. If this function is enabled, the system performance may be affected.

enable_ai_watchdog_healing

Parameter description: Enables or disables the self-healing function of the AI watchdog.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on**: enabled.
- **off**: disabled.

Default value: **on**

ai_watchdog_max_cpu_usage

Parameter description: Specifies the expected upper limit of the database CPU usage. The value is normalized based on the multi-core situation. If this parameter is set to **0**, the system does not check the CPU usage.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a floating-point number ranging from 0 to 1.

Default value: **0.8**

ai_watchdog_oom_dynamic_used_threshold

Parameter description: Specifies the expected upper limit of the dynamic memory usage of the database.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a floating-point number ranging from 0 to 1.

Default value: 0.95

ai_watchdog_oom_growth_confidence

Parameter description: Specifies the confidence level of the OOM detection algorithm.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a floating-point number ranging from 0.1 to 1.

Default value: 0.95

ai_watchdog_oom_malloc_failures

Parameter description: Specifies the maximum number of consecutive memory allocation failures tolerated. If the number of consecutive memory allocation failures exceeds this value, the OOM detection function may be triggered.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to 32000.

Default value: **50**

ai_watchdog_oom_other_used_memory_threshold

Parameter description: Specifies the expected upper limit of memory usage of other parts of the database, in MB.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 1 to 1048576.

Default value: 20480

ai_watchdog_oom_process_threshold

Parameter description: Specifies the expected percentage of the database process usage to the value of **max_process_memory**. When the threshold is reached, memory leakage determination is triggered. The value can be greater than 1.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a floating-point number ranging from 0 to 10.

Default value: 1.1

ai_watchdog_oom_shared_threshold

Parameter description: Specifies the expected upper limit of the shared memory usage of the database.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a floating-point number ranging from 0 to 1.

Default value: 0.4

ai_watchdog_rto_restriction_time

Parameter description: Specifies the RTO threshold of the AI watchdog self-healing function. If the RTO threshold is exceeded, self-healing is not performed. The unit is second.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 36000.

Default value: **600**

ai_watchdog_tolerance_times

Parameter description: Specifies the maximum number of consecutive abnormal events that can be tolerated by the AI watchdog before self-healing is started. This parameter can be used to avoid incorrect operations.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 100.

Default value: 4

ai_watchdog_tps_threshold

Parameter description: Specifies the lower limit of the expected TPS usage of the database instance. If the TPS usage is lower than the value of this parameter, the exception determination logic is triggered.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 32000.

Default value: **2**

ai_watchdog_wait_time

Parameter description: Adjusts the waiting time, in seconds. To prevent the database from frequently performing self-healing operations, the database waits for a period of time after startup.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 36000.

Default value: **1800**

ai_watchdog_warning_retention

Parameter description: Specifies the maximum number of alarm records that the AI watchdog can retain in the dbe_perf.ai_watchdog_detection_warnings view.

This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: an integer ranging from 0 to 32000.

Default value: 20

14.3.41 Global SysCache Parameters

enable_global_syscache

Parameter description: Specifies whether to enable the global system cache function.

Parameter type: Boolean

Unit: none

Value range:

- **on** indicates that the global system cache function is enabled.
- **off** indicates that the global system cache function is disabled.

Default value: **on**

Setting method: This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Enable this function to reduce the memory usage of the system cache and improve the concurrent expansion capability. You are advised to use this parameter together with the thread pool parameter. After this parameter is enabled, you are advised to set **wal_level** of the standby node to **hot_standby** or higher if you need to access the standby node.

global_syscache_threshold

Parameter description: Specifies the maximum memory usage of the global system cache. To use this parameter, you need to enable the **enable_global_syscache** parameter.

Parameter type: integer

Unit: KB

Value range: 16384 to 1073741824

Default value: **163840** (196-core CPU/1536 GB memory, 128-core CPU/1024 GB memory, 104-core CPU/1024 GB memory, 96-core CPU/1024 GB memory, 96-core CPU/768 GB memory, 80-core CPU/640 GB memory, 64-core CPU/512 GB memory, 60-core CPU/480 GB memory, 32-core CPU/256 GB memory, 16-core CPU/128 GB memory, 8-core CPU/64 GB memory, 4-core CPU/32 GB memory); **65536** (4-core CPU/16 GB memory)

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: MIN(Number of hot databases,Number of threads) x Memory size allocated to each database.

That is, **global_syscache_threshold** = **min(count(hot dbs), count(threads))** x **memofdb**.

The number of hot databases refers to the number of frequently accessed databases. In thread pool mode, the number of threads is the sum of the number of threads in the thread pool and the number of background threads. In non-thread pool mode, the number of hot databases is used.

memofdb indicates the average memory allocated to each database. The background noise memory of each database is 2 MB. Each time a table or index is added, 11 KB memory is added.

If this parameter is set to a small value, memory is frequently evicted, and a large number of memory fragments cannot be recycled. As a result, memory control fails.

14.3.42 Restoring Data on the Standby Node

standby_page_repair

Parameter description: Specifies whether to enable automatic page repair during replay on the standby node. In the current version, only the CRC check failure type can be repaired. Bad blocks in hash bucket tables, heap table FSM files, and VM files cannot be repaired. This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that the standby node automatically detects and repairs pages during replay.
- **off** indicates that the standby node does not automatically detect and repair pages during replay.

Default value: **on**

CAUTION

If a large number of physical bad blocks are continuously injected to the standby node, the replay performance of the standby node is affected, and Xlogs may be stacked on the standby node.

14.3.43 Delimiter

delimiter_name

Parameter description: Saves the name of a delimiter.

When the gsql client identifies delimiters, it immediately sends one or more input SQL statements to the server for execution. When there are many input statements and semicolons (;) exist in the statements, you can specify a special

symbol as the delimiter. This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

This parameter can be set only on the gsql client by running the **DELIMITER** command.

Value range: a string, consisting of one or more characters.

Default value: **";"**

14.3.44 Global PL/SQL Cache Parameters

enable_global_plsqlcache

Parameter description: Specifies whether to globally cache compilation products of packages, stored procedures, and functions, and cache execution products at the session level. Enabling this function can reduce the memory usage of database nodes in high concurrency scenarios.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: Boolean

- **on** indicates that compilation products are cached globally.
- **off** indicates that global cache is not performed.

Default value: **on**

max_execute_functions

Parameter description: Specifies the number of execution products of stored procedures and functions in a session. This parameter must be set when **enable_global_plsqlcache** is set to **on**. Otherwise, the setting is invalid.

If the number of execution products is greater than the value of **max_execute_functions**, only the most recently invoked execution products (the number is specified by **max_execute_functions**) are retained and others are cleared.

This is a POSTMASTER parameter. Set it based on instructions provided in **Table 14-1**.

Value range: a value ranging from 1 to 2147483647.

Default value: **1000**

max_compile_packages

Parameter description: Specifies the maximum number of package compilation results stored in the server.

Parameter type: integer

Unit: none

Value range: 0 to 2147483647. The value **0** indicates that this function is disabled and the number of package compilation products is not controlled. The default

value of this parameter can be adjusted based on different specifications. The recommended value can be obtained by rounding down the value of (**max_process_memory** x 2%)/4.4, in MB.

The value 4.4 MB is the average value obtained based on the simulation statistics in the lab. You need to observe the value in the actual scenario. If the value does not comply with the actual scenario, you need to adjust the value to adjust the memory usage of the stored procedure.

Default value:

Default values for different specifications are calculated by rounding down (**max_process_memory** x 2%)/4.4, in MB.

Setting method: This is a SUSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Storing too many package compilation results may occupy a large amount of memory. Setting this parameter to a proper value helps reduce memory usage and improve system performance.

14.3.45 Ledger Database

enable_ledger

Parameter description: Specifies whether to enable the ledger database. If this parameter is set to **on**, the ledger database is enabled, and a new tamper-proof mode can be created and the common mode can be changed to the tamper-proof mode.

Parameter type: Boolean

Unit: none

Value range:

- **on** indicates that the ledger database is enabled.
- **off** indicates that the ledger database is disabled.

Default value: **off**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: The ledger database is disabled by default. To enable it, set **enable_ledger** to **on**.

ledger_hist_level

Parameter description: Specifies whether to record SQL statements in the global blockchain table.

Parameter type: int

Unit: none

Value range:

- **0**: SQL statements are not recorded in the global blockchain table.

- **1**: SQL statements are recorded in the global blockchain table.

Default value: **1**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: Retain the default value **1**.

14.3.46 Creating an Index Online

delete_cctmp_table

Parameter description: Specifies whether to delete temporary tables generated during Ustore online index creation or rebuilding.

Parameter type: Boolean

Unit: none

Value range:

- **on**: indicates that the temporary tables are deleted.
- **off**: indicates that the temporary tables are retained.

Default value: **on**

Setting method: This is a USERSET parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: This parameter is enabled by default. Set this parameter to **off** if you want to retain temporary tables generated during online index creation or rebuilding.

14.3.47 Data Lifecycle Management: OLTP Table Compression

enable_ilm

Parameter description: Specifies whether to enable the OLTP table compression of data lifecycle management.

Parameter type: Boolean

Unit: none

Value range:

- **on**: enabled.
- **off**: disabled

Default value: **off**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestions: The default value is **off**. To enable this feature, contact Huawei engineers to purchase a license and then set this parameter to **on**.

14.3.48 Vector Database Parameters

maintenance_work_mem

Parameter description: Specifies the maximum amount of memory used in maintenance operations.

Parameter type: numeric

Unit: KB

Value range: [1024,2147483647] or [1MB, 2048GB)

Default value: **65536**/**64MB**

Level: session level

diskann_probe_ncandidates

Parameter description: Specifies the size of the candidate set when the gsdiskann index is used to retrieve vectors.

Parameter type: numeric

Unit: none

Value range: [1,32768]

Default value: **128**

Level: session level

NOTE

The **diskann_probe_ncandidates** parameter can be set and take effect for some queries that use the gsdiskann index. If this parameter is set to a large value, the query performance deteriorates. If this parameter is set to a small value, the recall rate is low. The recommended value is **128**. You are advised to obtain the optimal parameter configuration through experiments.

gsivfflat_probes

Parameter description: Specifies the number of inverted tables to be searched. If it exceeds the total number of inverted tables, the entire table is searched.

Parameter type: numeric

Unit: none

Value range: [1,32768]

Default value: **5**

Level: session level

NOTE

A larger value of **gsivfflat_probes** indicates a longer search time but more accurate results. You are advised to set **gsivfflat_probes** to 3% the value of **nlist** during index creation. You are advised to obtain the optimal parameter configuration through experiments.

gsivfflat_secondary_probes

Parameter description: Specifies the number of level-2 inverted tables to be searched. If it exceeds the total number of level-2 inverted tables, the entire table is searched.

Parameter type: numeric

Unit: none

Value range: [1,32768]

Default value: **5**

Level: session level

NOTE

With the same effect as that of **gsivfflat_probes**, **gsivfflat_secondary_probes** takes effect only when the vector index is a double-layer index. This parameter can effectively accelerate the query speed. You are advised to set it to a value ranging from 1/4 to 1/2 of **ivf_nlist2**. You are advised to obtain the optimal parameter configuration through experiments.

enable_vectordb

Parameter description: Specifies whether vector indexes can be created and whether vector indexes can be added, modified, and queried.

Parameter type: Boolean.

Unit: none

Value range:

- **on**: enabled.
- **off**: disabled.

Default value: **on**

Level: Global parameter (SIGHUP).

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Table 14-31 GUC parameters related to the vector database

GUC Parameter	Level	Value Range/Default Value	Description
maintenance_work_mem	Session	[1024,2147483647]/[1MB,2048GB) (**65536**/**64MB**)	Maximum amount of memory used in maintenance operations. The default unit is KB.

GUC Parameter	Level	Value Range/Default Value	Description
diskann_probe_ncandidates	Session	[1, 32768](128)	Size of the candidate set when the gsdiskann index is used to retrieve vectors.
gsivfflat_probes	Session	[1, 32768](5)	Number of inverted tables to be searched. If it exceeds the total number of inverted tables, the entire table is searched.
gsivfflat_secondary_probes	Session	[1, 32768](5)	Number of level-2 inverted tables to be searched. If it exceeds the total number of level-2 inverted tables, the entire table is searched.
enable_vectordb	Global parameter (SIGHUP)	[off, on](on)	An advance feature that specifies whether vector indexes can be created and whether vector indexes can be added, modified, and queried.

14.3.49 Enhanced TOAST

enable_enhance_toast_table

Parameter description: Specifies whether the enhanced TOAST out-of-line storage table is used.

Parameter type: Boolean

Unit: none

Value range:

- **on**: enabled.
- **off**: disabled.

Default value: **on**

Setting method: This is a SIGHUP parameter. Set it based on instructions provided in **Table 14-1**.

Setting suggestion: The default value of this parameter is **on**. To disable the enhanced TOAST out-of-line storage function, set this parameter to **off**.

14.3.50 Reserved Parameters

NOTE

The parameters below are reserved and do not take effect in this version.

acce_min_datasize_per_thread

enable_constraint_optimization

enable_hadoop_env

enable_hdfs_predicate_pushdown

enable_orc_cache

schedule_splits_threshold

backend_version

undo_zone_count

version_retention_age

max_que_size

Discarded Parameters

max_query_retry_times

tde_cmk_id

transparent_encrypted_string

transparent_encrypt_kms_url

transparent_encrypt_kms_region

hll_default_regwidth

hll_default_expthresh

hll_default_sparseon

hll_max_sparse

enable_compress_hll

time_to_target_rpo

15 FAQ

15.1 What is the maximum number of columns in a single GaussDB table?

Answer: 1600. The value varies according to the column type. The column type is not verified during table creation but is verified during data storage. For example, for a column of the bigint type, each column stores 8-byte data. If there are 1600 columns, 12800 bytes need to be stored, which exceeds 8 KB on a page, and an error is reported during insertion.

15.2 How do I query the partition and index information of a partitioned table?

Answer: You can use either of the following methods:

- You can use the pg_get_tabledef() function to view the definition of a table. The returned information includes table creation SQL statements, comments, indexes, and constraints.

  Example:
  ```
  gaussdb=#SELECT pg_get_tabledef('table_name');
  ```
- You can query the pg_partition view for partition information and the pg_indexes view for index information.

15.3 What is OID?

Answer: Object identifier (OID) is the unique identifier of a database object, which can be a database, table, index, or view.

15.4 What is UDF?

Answer: You can define a function and embed the user-defined function (UDF) in GaussDB to implement specific functions to meet different service scenarios.

15.5 What wildcards are supported in GaussDB? How do I use them?

Answer: GaussDB supports the following three types of wildcards:

- %: indicates any number of characters, including 0. It is used in the LIKE and NOT LIKE statements.
- _: indicates a character, which is used in LIKE and NOT LIKE statements.
- *: indicates any number of characters, including 0. It is used in some meta-commands.

  Examples:

```
-- Use wildcards to indicate any number of characters and query the tbl_test table for the data that starts with col1 and ends with ab.
gaussdb=#SELECT * FROM tbl_test WHERE col1 LIKE 'ab%';

-- Use a wildcard to query the data of any single character string starting with a and ending with b in the col1 column of the tbl_test table.
gaussdb=#SELECT * FROM tbl_test WHERE col1 LIKE 'a_b';

-- Query all tables whose names start with tbl.
gaussdb=#\dt tbl*
```

15.6 Is there a limit on the length of a database object name?

Answer: When a database object is created, the name cannot exceed 63 bytes. If the name exceeds 63 bytes, the database truncates the last byte (not the character). As a result, half a character may appear.

15.7 How do I view the creation time of a table?

Answer: See the value of the **created** column in the PG_TABLES system view.

```
-- Create a table.
gaussdb=#CREATE TABLE test(id int, name varchar(10));

-- Query the time when the test table is created.
gaussdb=#SELECT tablename,created FROM pg_tables WHERE tablename = 'test';
 tablename |              created
-----------+-------------------------------
 test      | 2024-01-12 14:50:59.611988+08
(1 row)
```

15.8 How do I create indexes in parallel?

Answer: Refer to the following method:

```
-- Set maintenance_work_mem based on the actual situation.
gaussdb=#SET maintenance_work_mem = '8GB';

-- Create a table.
gaussdb=#CREATE TABLE table_name (col1 int, col2 int);
```



```
-- Change the number of threads for creating indexes for a table based on the actual situation.
gaussdb=#ALTER TABLE table_name SET (parallel_workers=4);

-- Create an index.
gaussdb=#CREATE INDEX index_name ON table_name(col1);

-- Reset the parallel_workers parameter for the table.
gaussdb=#ALTER TABLE table_name RESET (parallel_workers);

-- Delete the index.
DROP INDEX index_name;

-- Drop the table.
gaussdb=#DROP TABLE table_name;
```

15.9 How do I create an auto-increment column?

Answer: GaussDB supports the creation of auto-increment columns. You can specify the SERIAL data type when creating a table.

Example:

```
gaussdb=#CREATE TABLE table_name(id serial, name varchar(20));
```

You can also use the following method:

```
-- Create a sequence.
gaussdb=#CREATE SEQUENCE tbl_person_id_seq;
-- Create the tbl_persion table. The value of the id column is automatically increased based on the sequence specified by tbl_person_id_seq.
gaussdb=#CREATE TABLE tbl_persion(
    id int NOT NULL DEFAULT nextval('tbl_person_id_seq'::regclass),
    name varchar(20));
```

15.10 Can I query the GaussDB memory usage through SQL statements?

Answer: You can query the context name, level, total size, and available size of the shared memory in the pg_shared_memory_detail view.

```
gaussdb=#SELECT * FROM pg_shared_memory_detail;
```

15.11 What are the differences between LIMIT 2, LIMIT 2,3 and LIMIT 2 OFFSET 3?

Answer: LIMIT and OFFSET are keywords used to limit the number of query results returned in the SELECT statement.

- **LIMIT 2**: Only the first two rows of data are returned.
- **LIMIT 2,3**: A total of three rows of data from the third row are returned.
- **LIMIT 2 OFFSET 3**: A total of two rows of data from the fourth row are returned.

15.12 How do I create a column whose default value is the current time?

Answer: When creating a table, set the default value of the column to **CURRENT_TIMESTAMP**.

```
gaussdb=#CREATE TABLE tbl (id int, modtime date DEFAULT CURRENT_TIMESTAMP);
```

15.13 How do I determine whether a column is null?

Answer: Use IS NUL and IS NOT NULL to determine whether a column is null. For example:

```
gaussdb=#SELECT * FROM tab WHERE col1 IS NULL;
```

15.14 How do I obtain the username for connecting to a database?

Answer: Run the following command to query the current username:

```
gaussdb=#SELECT current_user;
```

15.15 How do I query the time difference between two time points?

Answer: You can use the following method to calculate the time difference:

- Use the age() function to calculate the time difference between two time points.

```
gaussdb=#SELECT age(timestamp '2001-04-10 14:00:00', timestamp '2001-04-06 13:00:00');
       age
-----------------
 4 days 01:00:00
(1 row)
```

- Convert strings to the date type and then subtract them.

```
gaussdb=#SELECT ('2001-04-10 14:00:00'::date - '2001-04-06 13:00:00'::date);
 ?column?
----------
        4
(1 row)
```

- Use the date_part() function to obtain the value of the subdomain in a date or time value.

```
gaussdb=#SELECT date_part('day', '2001-04-10 14:00:00'::timestamp - '2001-04-06
13:00:00'::timestamp);
 date_part
-----------
         4
(1 row)
```

15.16 What are the types of SQL languages?

Answer: SQL languages are classified into the following types:

- Data definition language (DDL) is used to define or modify an object in a database, such as a table, an index, or a view.
- Data manipulation language (DML) is used to perform operations on data in database tables, such as inserting, updating, querying, or deleting data.
- Data control language (DCL) is used to set or change the permissions of database users or roles.

15.17 What is the function of a trigger?

Answer: A trigger is a special stored procedure that is triggered and executed by events.

- Constraints can be strengthened to maintain data integrity and consistency.
- Operations in the database can be traced. Unauthorized updates and changes are not allowed.
- Concatenated operations can be performed.

15.18 What are the four characteristics of correctly executing database transactions?

Answer: During data write or update, the database management system (DBMS) must meet the following requirements: Atomicity, Consistency, Isolation, and Durability (ACID) to ensure that transactions are correctly executed.

- Atomicity: also called indivisibility. All operations in a transaction are either completed or not completed, and do not end in an intermediate phase. If an error occurs during transaction execution, the transaction is rolled back to the state before the transaction starts, just as if the transaction has never been executed.
- Consistency: The database integrity is not damaged before and after a transaction starts. This means that the written data must fully comply with all preset rules, including the accuracy of the data, the serialization of the data, and the spontaneous completion of the scheduled work by the subsequent database.
- Isolation: also called independence. Transactions are often executed concurrently. Isolation ensures that concurrent execution of transactions leaves the database in the same state that would have been obtained if the transactions were executed sequentially. Transaction isolation is divided into different levels, including read uncommitted, read committed, repeatable read, and serializable.
- Durability: Once a transaction has been committed, it will remain committed even in the case of a system failure.

15.19 What are the differences between the DROP, TRUNCATE, and DELETE methods in GaussDB?

Answer: The differences between the DROP, TRUNCATE, and DELETE methods lie in the deletion speed and scope. The details are as follows:

- The DROP statement can be used to delete an entire table, including the table structure, data, indexes, and permissions.
- The deletion speed of the TRUNCATE statement is medium. The TRUNCATE statement can delete all data in a table but does not delete the table structure.
- The deletion speed of the DELETE statement is the slowest. You can delete data from a table based on conditions, excluding the table structure.

15.20 How many bytes does a Chinese character occupy in GaussDB?

Answer: It depends on the character set encoding of the database. For the GBK character set, a Chinese character occupies two bytes. For the UTF8 character set, a Chinese character occupies three bytes.

